TAMPERE UNIVERSITY OF TECHNOLOGY

OLLI SUOMINEN

TRANSFORM-BASED METHODS FOR STEREO MATCHING AND DENSE DEPTH ESTIMATION

Master of Science Thesis

Examiners:
Dr. Atanas Gotchev
Dr. Miska Hannuksela
Examiner and topic approved by
the Faculty Council of the Faculty of
Computing and Electrical Engineering
on 4 April 2012

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
OLLI SUOMINEN : Transform-based methods for stereo matching and dense depth estimation
Master of Science Thesis, 50 pages, 2 Appendix pages
May 2012
Major: Signal Processing
Examiners: Dr. Atanas Gotchev, Dr. Miska Hannuksela
Keywords: stereo matching, transform domain, sign-only correlation, DCT, Haar

Stereo matching is a passive method for estimating depth of a scene from two views from different perspectives. Parallax creates a disparity between the relative positions of scene points on the imaging planes depending on the distance of the points. The principle of stereo matching is to extract those disparities by finding the corresponding points between the images. Although stereo matching has been extensively studied, the existing solutions are still compromises between computational load and achieved quality. In this thesis, advances are made on both fronts.

At the core of the matching algorithm is the similarity measure, which directly determines how well correspondences are found and how reliable they are. Traditionally, matching has been done in spatial domain using pixel differences such as sum of absolute differences (SAD). In this thesis, a similarity measure is proposed for use in stereo matching that is based on analysis of coefficient signs of transform domain representations. While originally formulated as an extension of Fourier domain phase-only correlation to the discrete cosine transform (DCT), here the method is developed further by applying it to a number of real-valued abstract harmonic transforms, including type II DCT, integer DCT, Walsh-Hadamard and a modified version of Haar. Results are presented showing that the method in general provides better quality than the reference algorithm SAD, while Haar is shown to be the best performing transform, both in terms of quality and speed.

Furthermore, the approach is adapted to a mobile platform by replacing the transform with an even simpler one, the census transform. An efficient implementation is developed, which utilizes the single instruction, multiple data (SIMD) enabled NEON core included in many ARM processors currently dominating the mobile market. Special attention is paid to the alternate methods of performing a population count on a variable, which is a key component in computing the similarities. Subjective testing along with numerical measurements set the census-based matching slightly under the reference point SAD in terms of quality, but speed-wise SAD is clearly out-performed by the census approach, thus establishing it as a competitive candidate for stereo matching in mobile applications.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
OLLI SUOMINEN: Muunnoksiin perustuvat menetelmät stereovastaavuuksien etsimiseen ja tiheän syvyyskartan estimointiin
Diplomityö, 50 sivua, 2 liitesivua
Toukokuu 2012
Pääaine: Signaalinkäsittely
Tarkastajat: Atanas Gotchev, Miska Hannuksela
Avainsanat: stereovastaavuus, muunnostaso, DCT, Haar

Stereovastaavuuksien etsiminen on passiivinen menetelmä syvyyden estimointiin kohteesta eri perspektiiveistä otettujen kuvien perusteella. Parallaksin takia kohteen pisteiden projektiot ovat kuvatasoilla suhteessa eri asemassa toisiinsa nähden riippuen pisteiden etäisyydestä kuvatasosta. Etsimällä eri kuvista vastaavat pisteet voidaan niiden sijaintien erotuksen perusteella määrittää pisteen etäisyys. Vaikka tähän periaatteeseen perustuvia menetelmiä on tutkittu laajalti, olemassaolevat ratkaisut ovat kuitenkin kompromisseja laskennallisen kuorman ja saavutetun laadun välillä. Tässä opinnäytetyössä parannusta esitetään molempiin osa-alueisiin.

Vastaavuuksien etsiminen perustuu oleellisesti samankaltaisuuden määritelmään. Se määrittää, kuinka hyvin vastaavuuksia löydetään ja kuinka luotettavia ne ovat. Perinteisesti vastaavuushakua on tehty tilatasossa vertailemalla pikselien intensiteettejä, kuten absoluuttisten erotusten summa -tekniikassa. Vastaavuuksia voidaan kuitenkin etsiä tutkimalla kertoimien etumerkkejä muunnostasossa. Tekniikka perustuu vaiheiden korrelaatioon Fourier-tasossa, ja tämän ominaisuuden ulottumiseen koskemaan myös diskreettiä kosinimuunnosta (DCT). Tässä työssä tekniikkaa laajennetaan koskemaan useita realiarvoisia, abstrakteja, harmonisia muunnoksia, kuten tyypin II DCT, kokonaisluku-DCT, Walsh-Hadamard ja muunneltu Haar. Kokeelliset tulokset osoittavat, että yleisesti ottaen menetelmä toimii paremmin kuin vertailualgoritmina käytetty SAD. Kokeilluista muunnoksista Haar paitsi tuottaa parhaita tuloksia, on myös yksinkertaisin laskea.

Menetelmää sovelletaan myös mobiilialustalle, jolloin käytettyä muunnosta yksinkertaistetaan entisestään korvaamalla se ns. census-muunnoksella. Tätä varten työssä kehitettiin tehokas toteutus, joka käyttää mobiilimarkkinoita hallitsevan ARM-prosessorin rinnakkaissuorituslaajennusta, NEON-ydintä. Erityishuomiota kiinnitetään ns. population count -operaation toteutukseen, joka on oleellinen osa algoritmin tehokasta suoritusta. Käyttäjäkokeiden ja objektiivisten mittausten mukaan censuksen tuottama laatu jää jälkeen hieman SAD:n tuottamasta, mutta nopeuden suhteen census-pohjainen menetelmä on selkeästi kilpailijaansa parempi, ja näin ollen kilpailukykyinen vaihtoehto vastaavuushakujen toteutukseen mobiilialustoilla.

PREFACE

The research for this thesis was done as a part of the 3D video capture branch of the Advanced Video and Image Processing Algorithms project, a collaboration between the Department of Signal Processing at Tampere University of Technology and Nokia Research Center. The aim of the work was to explore fast approaches for estimating depth from stereo images and implementing such algorithms on a mobile platform.

I would like to thank my supervisor Atanas Gotchev for the feedback and guidance during the thesis work. I am also grateful for the support from the rest of the 3D media team at TUT, especially Atanas Boev, Mihail Georgiev and Sergey Smirnov. They have all helped in creating a flexible environment for combining my studies with the research activities of the group, enabling me to finish this thesis and graduate as scheduled, while still having two years worth of valuable research experience.

From NRC, I would like to thank Miska Hannuksela for the opportunity to get involved in this kind of work, and those who went through the trouble of helping me settle in during the period of working in their premises.

A special thanks belongs to Heikki Huttunen, who has affected the path towards the completion of this thesis on several occasions. From the face tracking demo, which cemented my intention of taking up signal processing, to introducing me to the people in the 3D team for my BSc thesis and to organizing the journal club where a presentation inspired the topic of this thesis, it is safe to say things would not have turned out the way they did without him.

Tampere, 22nd of May 2012

Olli Suominen

CONTENTS

1. Introduction 1
2. Theoretical background 4
2.1 Stereo matching pipeline 4
2.1.1 Rectification of stereo images 4
2.1.2 Similarity measures 5
2.1.3 Cost volume aggregation 6
2.1.4 Disparity computation and confidence 9
2.1.5 Post processing 10
2.1.6 Left-right consistency 12
2.2 Transform-based matching 12
2.2.1 Transforms 13
2.2.2 Phase-only and sign-only correlation 17
2.2.3 Computing similarity between transformed windows 19
3. Implementation 21
3.1 Generalized transform-based matching 21
3.2 Census-based matching for mobile 24
3.2.1 Implementation platform 25
3.2.2 Optimization strategies for NEON 25
3.2.3 Implementation specifics 27
4. Results and evaluation 31
4.1 Generalized transform-based methods 32
4.2 Census-based mobile implementation 38
4.2.1 Performance analysis 38
4.2.2 Subjective testing 41
5. Conclusions 44
References 47
APPENDIX 1: Aggregated disparity maps 51
APPENDIX 2: Example data: varying exposure and illumination 52

TERMS AND SYMBOLS

$\mathbf{C}, \mathbf{C_A}$	Cost volume, a 3-dimensional structure holding results of evaluated similarity measures, the $_A$ marking that aggregation has been performed on it
$\mathbf{D}$	Disparity map showing disparity for each point
$\mathbf{d_{min}}, \mathbf{d_{max}}$	Minimum and maximum disparity values expected to be in the scene, i.e. limits of the disparity search range
$\mathbf{f}, \mathbf{g}$	Real valued 2D arrays, i.e. images
$\mathbf{F_T}, \mathbf{G_T}$	Representations of f and g in transform domain reached by applying transform T
$\mathbf{L}, \mathbf{R}$	Left and right source images
$\mathbf{s_T}(\mathbf{f}, \mathbf{g})$	Similarity between f and g in the transform domain T
$\mathbf{T}$	Without ambiguity, denotes either the transformation matrix or the 2-dimensional transform itself
Aggregation	Performed on a cost volume to enforce spatial correlation
Bitstring	A compressed representation of a transformed window where each bit corresponds to a pixel
Cost	Here, a measure of the difference between image blocks
Disparity	Horizontal displacement between the projections of a point in two images from different viewpoints
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
MSE	Mean Squared Error
NEON	The SIMD core accompanying an ARM processor
Population count	An operation that counts the bits set to one in a variable
PBP	Percentage of bad pixels, i.e. those pixels that do not match the ground truth
Reference	A block for which a corresponding block is searched for
Similarity	Conceptually the inverse of cost, here a measure for how similar blocks of images are
SIMD	Single instruction, multiple data - multiple inputs are processed with the same operation in parallel
SAD	Sum of Absolute Differences, the stereo matching algorithm used as reference
Target	A candidate block that is compared to the reference to find the best match

1. INTRODUCTION

Extracting geometry from captured scenes has long been an area of interest for machine vision [1] . More recent developments have made it relevant also for 3D media applications [2]. A common format for this kind of geometric information captured from a certain angle is the depth map. If interpreted as an image, each pixel value of a depth map tells a distance between the capture device and a point in the scene. Transparent objects aside, this is enough to contain all the information that can practically be acquired from a certain viewpoint. A specific need in 3D media is to create a presentation for 3D video called view+depth. It is used for creating novel views to fit different types and sizes of stereoscopic and multiview displays. [3]

Different methods exists for the extraction of scene geometry as a depth map. Active methods include using time-of-flight cameras, which measure the scene by sending infrared pulses into it [4], and structured light approaches, such as the Microsoft Kinect, which analyze the behavior of some known pattern projected onto the scene [5]. Passive methods do not require anything to be projected or transmitted separately, but they measure the reflections of ambient conditions. One such passive method is stereo matching, which is based on using some kind of a similarity measure for finding correspondences between several captures of a scene from different perspectives.

Given two images taken from the same scene, but from different viewpoints, stereo matching aims to extract a depth map. The key principle is that of parallax, demonstrated in Figure 1.1 - objects further away are projected onto the imaging planes of the two cameras relatively closer to each other than objects nearby. Ideally, the distance between objects on the imaging planes (i.e. the stereo disparity between the objects) defines exactly their distance from the cameras. While the human eye is skilled in identifying similar points between the images, for a computer the task is significantly more difficult. There is a number of problems from how to formulate the concept of similarity to the machine, to how to deal with areas lacking enough information to make a proper evaluation.

While stereo matching has been the target of extensive research over the years, a general solution has not yet been found. The approaches can be divided into two parts. Local matching is based on finding similarities between the spatial neighbor-

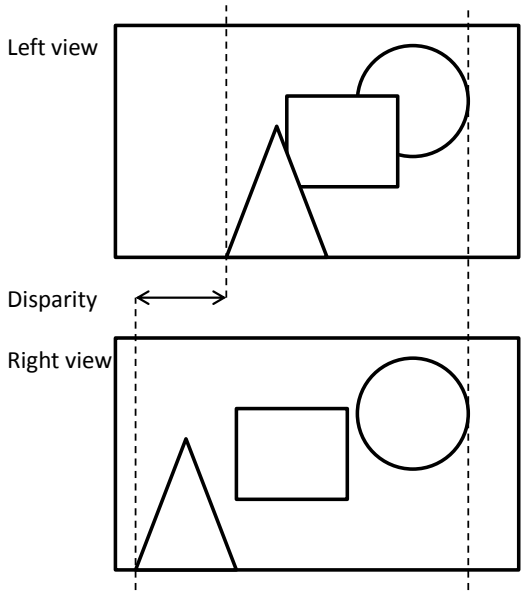


Figure 1.1: A stereo pair of a a scene consisting of three objects, a triangle in the front, a box in the middle and a sphere in the back. Due to parallax, the triangle has a disparity between the two images, whereas the sphere is too far for any significant difference to be detected. Stereo matching aims to discover this disparity in order to determine the distance of the objects to the camera.

hoods of given points in a stereo pair. In contrast, global methods aim to optimize the disparity map in relation to constraints stemming from assumptions on the characteristics of smoothness and discontinuities of disparity maps. Roughly speaking, local methods are faster, but provide worse quality, whereas global methods tend to be slower and give better results. [6]

The focus of this thesis is to study and develop faster and more robust methods of making similarity comparisons. These methods are applied for local stereo matching in two complementary paths - a general approach for stereo matching in different transform domains, and a more practical implementation for a mobile device. The common principle for both is to extract a small memory footprint presentation of windowed image segments centered on the pixels of the images by applying a suitable transform, and the way the information is efficiently stored and processed internally by the algorithm. This presentation is then used for computing similarity between the segments, i.e. blocks. For the mobile implementation, the technique is in adapted from an existing matching method utilizing the census transform [7] to accommodate the stringent performance limitations of the implementation platform. The similar idea is used in a more generalized setting without any specific performance constraints by experimenting with different transforms.

The rest of the thesis is structured in the following way. The general principles and practices of stereo matching are described in chapter 2 together with the theoretical

background of the transform-based methods proposed for this application. The implementations for both the generalized and the mobile oriented approaches are described in detail in chapter 3. Experimental results on quality and computational performance for both are presented in chapter 4, including the results of a small scale subjective study. Finally, conclusions based on the results and on observations made during the implementation stage are done in chapter 5.

2. THEORETICAL BACKGROUND

Local stereo matching methods rely on comparing windowed sections of both stereo pairs in order to find the best match for each window, and therefore the displacement(i.e disparity), of those blocks in relation to the corresponding pair. The disparity values then correspond to the distances of those points from the camera. The actual disparity to depth mapping is dependent on the distance between the cameras and their intrinsic parameters. In 3D media applications, the depth is usually represented as 8-bit integer values, 255 marking objects close by and 0 objects far away from the camera, i.e. a depth map.

2.1 Stereo matching pipeline

Most modern stereo matching applications follow the structure described in [8] (Figure 2.1). Some similarity measure is first used to determine the similarity of a point in the other image to multiple candidate points in the other one. Cost volume aggregation is applied to leverage the assumption that points in the same neighborhood likely have similar disparities. The disparity estimate is then the disparity candidate which yields the highest similarity. However, the specific contents of each step varies between algorithms.

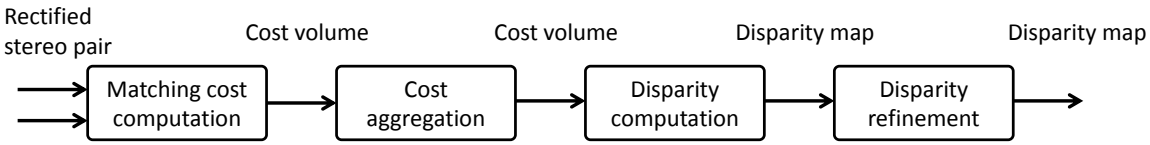


Figure 2.1: A common stereo matching pipeline

2.1.1 Rectification of stereo images

The majority of stereo matching algorithms assumes that the input stereo pair is rectified. A rectified stereo pair means that the normals of the image planes of both cameras are parallel and point towards the scene, and that a scene point is located on the same horizontal line in both images , i.e. an epipolar line[9]. This allows for the algorithm to search for stereo correspondences in one dimension, along the epipolar line. This drastically reduces the area that has to be covered in order to

perform an exhaustive search by removing one degree of freedom. It also helps in making correct matches as the similarity measure does not have to have scale- or orientation-invariant properties.

In order to perform rectification on a stereo pair, two sets of parameters, intrinsic and extrinsic, must be known [10]. Intrinsic parameters characterize the internal configuration of the camera: the focal length, effective pixel density, the optical axis and the skew of the sensor. Extrinsic parameters describe the physical arrangement of the cameras, specifically their rotation and translation. Additional problems are caused by the physical deficiencies of the lenses. Typical artifacts are tangential and radial distortions. Tangential distortions are caused by the misalignment of lenses in relation to each other, while radial distortions are created by the non-planar shape of the lens [9]. In practice, it is nearly impossible to set the capturing environment and the cameras up so precisely that all these parameters are consistent between the cameras.

Both of the parameter sets can be extracted for a certain camera configuration by capturing several different poses of a known pattern. A common pattern used for this is a checkerboard. By measuring the shape of the squares, it is possible to define how going through the camera system alters the pattern and then to compensate accordingly. The same pattern gives cues on how to correct the radial distortions. [11]

Rectification is performed after capture and before the images are input to the stereo matching algorithm. Based on the extracted calibration parameters, such a transformation is formed that when applied to the images, they fulfill the requirements of a rectified pair. Applying the transform to the regular grid of the image requires interpolation, as the resampling will in most cases require pixels to be acquired from between the existing samples. As with all resampling tasks, this should be performed in "reverse", going through grid of the the result image, interpolating a value for each pixel by sampling the existing image. Transforming the existing pixels into a new grid would only create a nonuniformly sampled, sparse representation.[10]

2.1.2 Similarity measures

The main factor in the outcome of the stereo matching is the choice of similarity measure, as different measures have different properties. The measures are computed between the left and right images L and R at discrete pixel coordinates (x, y) Common options are the Squared Intensity Difference,

$$C_{SID}(x, y) = (L(x, y) - R(x, y))^2 , \tag{2.1}$$

also known as Mean Squared Error, and Absolute Intensity Difference,

$$C_{AID}(x, y) = |L(x, y) - R(x, y)|, \tag{2.2}$$

a.k.a. Mean Absolute Difference. A wide array of different metrics have been suggested, all with their own benefits and drawbacks. Some metrics are insensitive to differences in gain and bias, such as the zero compensated AID [12] and census transform [7]. Census transform and its usage as a similarity measure is described in section 2.2.1. In this part of the stereo matching pipeline lies also the contribution of this thesis. The suggested similarity measure is described in detail in section 2.2.

When a similarity is tested for each candidate over the whole image and the whole search range, a single similarity can be seen as the value in a three dimensional structure $C(x, y, d)$, where x and y correspond to the spatial dimensions of the image where the reference block is taken from, and d is the disparity candidate. This structure is called a cost volume. The cost volume can be manipulated to improve the quality of the estimate. Often the metric and the cost aggregation are referred to in a single term, such as SAD, Sum of Absolute (intensity) Differences. SAD is used here as the reference algorithm when experimenting with the other metrics. For the remainder of the thesis, the term cost is used to refer to the similarity value such that a low cost implies a high similarity, and therefore a good match, and vice versa.

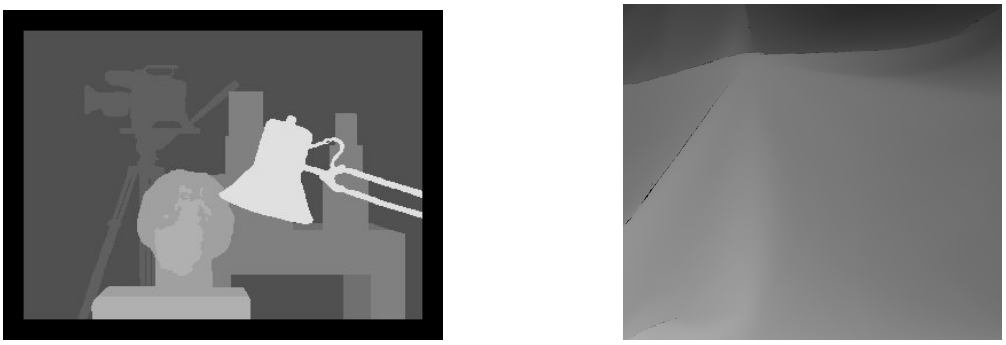
2.1.3 Cost volume aggregation

The assumption is that for natural scenes, neighboring disparity values are highly correlated, even more so than in a color image. I.e., disparity maps are assumed to be piecewise constant, or piecewise smooth, such as in Figure 2.2. To reduce the effect of erroneous matches and noise, the consistency of the cost volume can be reinforced by aggregating the cost values over a support region, which may be anything from one to all three dimensions. For instance, a straightforward way of doing this is to treat the xy-plane at each disparity d as an image, and to apply a spatial filter to it. The drawback with this is, that this effectively requires filtering as many images as there were disparity estimates in the search range. [10]

The method of aggregation can be of varying complexity. Quite a lot of different approaches have been suggested, which vary in the sizes and shapes of the support windows, and in the weights given to their contents [13]. An extremely simple and fast way is to use a box filter, which for a $N \times N, N = 2r$ filter window is

$$C_A(x, y) = \frac{1}{N^2} \sum_{n=-r}^{r} \sum_{m=-r}^{r} C(x + n, y + m). \tag{2.3}$$

which can be efficiently implemented using summed area tables (SAT) [14]. Fil-



a) b)

Figure 2.2: Two ground truth disparity maps showing a) piecewise constant b) piecewise smooth surfaces

tering with SAT is done in two steps. First, the values of the signal are summed along the rows and columns in to a lookup table S so that

$$S(x,y) = \sum_{n=0}^{x} \sum_{m=0}^{y} C(n,m). \tag{2.4}$$

A response exactly equivalent to 2.3 for the box filter at x, y is then computed using

$$\begin{aligned} C_A(x,y) = \frac{1}{N^2} \big(&S(x+r, y+r) - S(x-r-1, y+r) \\ &- S(x+r, y-r-1) + S(x-r-1, y-r-1)\big). \end{aligned} \tag{2.5}$$

The assumption of continuous depth does fail in object edges, which lead to disparity discontinuities. A large aggregation window with constant weights is more robust to matching errors, but is problematic in the edges and in fine details which are smaller than the window. This is why edge-aware filters for cost volume aggregation have been suggested. Those filters attempt to preserve the edges, and still make the allegedly uniform surfaces of objects consistent. One approach used for this is the joint bilateral filter [15], which can retrieve edge information from the corresponding image in the stereo pair, and use that to preserve the edges in the cost volume and thus the resulting disparity map. The bilateral filter takes into account both the intensity and location differences, weights them with some arbitrary functions and determines how much each pixel in the neighborhood contributes to

the filter response. The response of the basic version is

$$C_A(x,y) = \frac{\sum_{m=-r}^{r}\sum_{n=-r}^{r} f(m,n)g(C(x+n,y+m) - C(x,y))C(x+n,y+m)}{\sum_{m=-r}^{r}\sum_{n=-r}^{r} f(m,n)g(C(x+n,y+m), C(x,y))}, \quad (2.6)$$

where f is the weighting function for the distance and g is the weighting function for the difference in intensity. A common choice for both is the Gaussian function with different parameters,

$$f(\Delta x, \Delta y) = \exp\left(-\frac{1}{2}\frac{\sqrt{\Delta x^2 + \Delta y^2}}{\sigma_d}\right) \quad (2.7)$$

$$g(\Delta i) = \exp\left(-\frac{1}{2}\frac{\Delta i^2}{\sigma_i}\right), \quad (2.8)$$

where σ_d and σ_i are the parameters determining the steepness of the Gaussian fall-off for distance and intensity, respectively.[16] With low values, the contribution of different intensities and far away pixels becomes larger, blurring the image, while with high values the result is highly effected by noise and texture. Therefore the suitable parameters are a compromise of picking up on the actual edges instead of just any intensity changes, but not blurring too much to also loose the edges.

A stronger edge preserving effect is gained if it is assumed that intensity edges in the corresponding stereo image are also edges in disparity. Therefore a joint bilateral filter is applied, where the weighting function g gets its parameters from the color image, e.g. the right view R.

$$C_A(x,y) = \frac{\sum_{m=-r}^{r}\sum_{n=-r}^{r} f(m,n)g(R(x+n,y+m) - R(x,y))C(x+n,y+m)}{\sum_{m=-r}^{r}\sum_{n=-r}^{r} f(m,n)g(R(x+n,y+m) - R(x,y))}. \quad (2.9)$$

However, the naive implementation of a bilateral filter is computationally quite demanding. As the weights of the filter vary between each pixel, the same tricks that are used to perform efficient convolution cannot be directly applied. However, there are certain optimizations of the algorithm that improve the situation. A recent suggestion is to perform a decomposition of the filter into a series of linear filters, from which the response is then interpolated. This version can utilize the SAT approach for computing those linear sections. The performance gain comes from the fact that the amount of linear filters can be heavily quantized from e.g. 255 to

3 or 4 without much noticeable effect. [17]

Transform-based methods may not require a separate cost volume aggregation step, and do not benefit from it as much as those based on spatial domain differences. As the similarity is computed based on a windowed image segment, there is some amount of aggregation done already, although not strictly in the same manner as with separate cost volumes. This issue is discussed more in section 4.1 with some experimental data on the subject.

2.1.4 Disparity computation and confidence

For any coordinates (x, y), the corresponding section of a cost volume is a cost vector. Figure 2.3 displays examples of actual cost vectors. The first one is a good cost vector with a distinct minimum value, while the other is a vector from an area, which is not visible from both views, i.e. from an occluded pixel. No real match exists for such pixels.

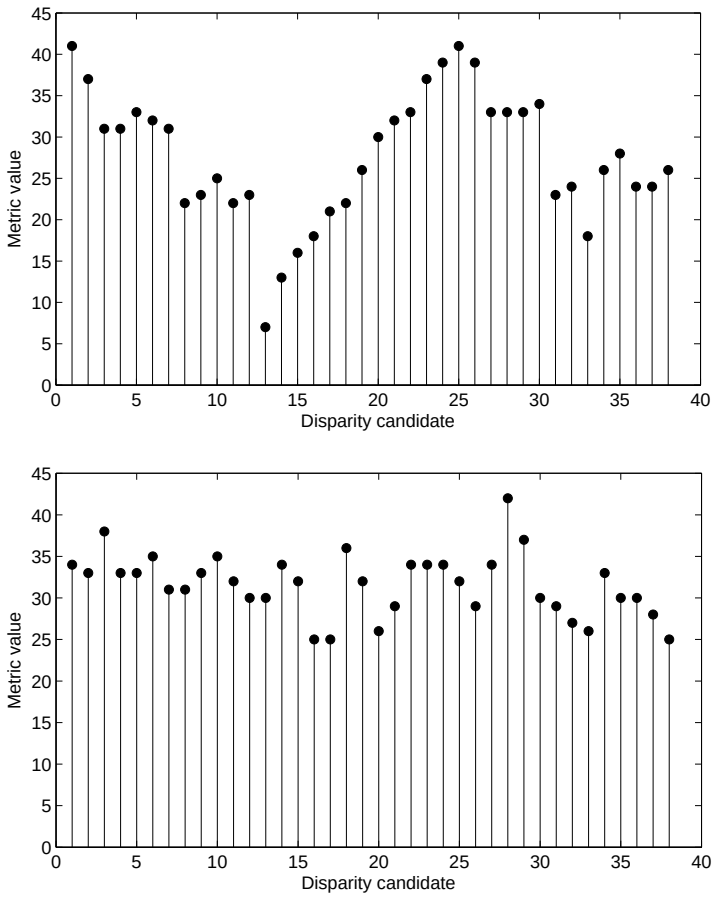


Figure 2.3: Two examples of cost vectors, where there is a distinctive minimum value corresponding to the disparity value at $x = 13$ (upper image) and where there is no clear choice due to the fact that the point is occluded from the viewpoint of the other image (lower image)

From the cost vector, retrieving the disparity estimate is trivial. A simple Winner-Takes-All (WTA) scheme is applied, where the lowest matching cost is selected as the disparity estimate.

$$D(x,y) = \underset{d\in[d_{min},d_{max}]}{\arg\min} \; C(x,y,d)), \tag{2.10}$$

If the similarity measure is a correlation or similar, the procedure is otherwise the same, but the maximum value is selected. The disparity estimate is not, however, the only information that can be extracted from the cost vector. For post processing, it would be beneficial to know, how trustworthy a single disparity estimate is, i.e. the *confidence* of the estimate. At its simplest, the cost can be used as a confidence value, with a better matching criteria also meaning a higher confidence. Another possibility that offers another interpretation of confidence is Peak to Peak Ratio, which measures how unique the match was. If the ratio is small, there is no "safety margin", suggesting that the match may have been ambiguous because some other disparities were almost equally good options. If the ratio is high, the disparity is a clear winner, and therefore likely to be reliable, i.e. has a high confidence. [8]

2.1.5 Post processing

From a filtering perspective, the wrong matches can be considered as noise. However, the type of noise is typically not normally distributed when using transform-based similarity measures. It appears to be more or less statistically independent from the true signal. Those kinds of metrics are slightly unpredictable in the sense that sometimes a single estimate is completely off the true disparity. Therefore it is not a valid option to simply smooth the disparity estimate with a FIR-filter such as Gaussian, as it would only spread the errors around, thus creating "bumps" in areas that should be planar. The mismatches mostly resemble salt&pepper noise, although depending on the implementation of the Winner Takes All -procedure, it tends to favor either high or low values of the tested disparity range.

The remedy to this kind of noise is to use rank-order filtering, of which median filtering is a special case. Depending on which end of the disparity range the WTA favors and if emphasizing the foreground or the background desired, the effect of the filter can be controlled by using other percentiles than the 50% of the median filter. This choice has a negligible effect on the computational performance.

Assuming a fixed range of values, the median filter (or any other rank order filter) can be implemented using histograms. This holds for a disparity map for obvious reasons, as the cameras capturing the stereo pair have a fixed relative distance, which limits the disparity range. Given the histogram $h_f(n)$ of a $N \times M$ image f, the median of f is such a k that satisfies

$$\sum_{i=1}^{k} h_f(i) = \frac{NM}{2}. \tag{2.11}$$

Assuming b, the number of evenly distributed bins in $h(n)$, is the same as $f_{max} - f_{min} + 1$, i.e. there is a bin for each value of the image, the output will be exact. If there are fewer bins, the output will be quantized. Computing a histogram is a distributive operation, and this can be exploited when computing histograms of spatially consecutive data in a sliding window manner. To ease demonstrating this property, the histogram operation $H(f)$ is defined as producing h_f, the histogram of f. Given two sets of values a and b,

$$H(a \cup b) = H(a) + H(b). \tag{2.12}$$

This property can be utilized both horizontally and vertically. For each column of the input image, one histogram is maintained. As the first step, a histogram of the first column segments has to be computed. Each of those histograms contains the information for a section of the column that is the height of the filtering window. The histogram for a 2D section can be computed as the sum of those individual, column-wise histograms. Furthermore, the histogram of a window differs from the window one sliding step before by the histograms of two columns, the one leaving the window and the one entering it. For adjacent columns $c_{x,y}$, whose first pixel is located at (x, y), and with a 8×8 window size

$$H(c_{1,y} \cup ... \cup c_{8,y}) = H(c_{0,y} \cup ... \cup c_{7,y}) - H(c_{0,y}) + H(c_{8,y}). \tag{2.13}$$

This allows the computation of the histogram for the next window with only $2b$ additions/subtractions, with the exception of the first window, which has to be summed from the columns individually. When at the end of the first horizontal pass, the column-wise histograms are updated. There is again a difference of two areas - the above pixel that is leaving the filter window, and the pixel below that is entering it,

$$H\big(c_{x,y}\big) = H\big(c_{x,y-1}\big) - H\big(f(x,y)\big) + H\big(f(x,y+8-1)\big). \tag{2.14}$$

Updating the column-wise histograms requires finding the correct bin for two values. For single valued bins, this is achieved by two indexing and two increment/decrement operations. Altogether, the distributive histogram offers a major improvement over the conventional method of computing histograms without the large memory footprint of alternative techniques. [18]

2.1.6 Left-right consistency

Disparity estimation can be done from the viewpoint of either of the images in the stereo pair. However, if it is done for both, the resulting two disparity maps can be used to detect erroneous matches from each other. Ideally, all the non-occluded areas of both disparity images should be consistent with each other. This assumption is utilized by performing a left-right consistency check. Disparity values are projected from e.g. the left disparity image to the right by what is essentially a form of image-based rendering. The values of the disparity image give the displacements for projecting that image itself. It is reasonable to allow a slight difference in the values as it hardly makes any difference in the outcome. If the projected disparity value differs significantly from the corresponding value in the other image at the projected coordinates, there is reason to suspect that the value might be wrong. [7] The check is done by evaluating

$$|D_R(x,y) - D_L(x + D_R(x,y), y)| > t \tag{2.15}$$

for each pixel, and if it holds, the pixel is marked as inconsistent. The amount of allowed variation is controlled by the parameter t. Often $t = 1$ is a reasonable choice. The primary cause of inconsistencies is occluded areas in the image, but this method can also catch other types of mismatches. There are various options on how to fill in those pixels that are evaluated as inconsistent. One can for instance replace the inconsistent value with the response of a median filter with selective sampling, i.e. compute the median only of those pixels that were found to be consistent.

2.2 Transform-based matching

When transforming images into the Fourier domain, there is a property called phase correlation that relates the shifting of the image in spatial domain to correlation of the phase components [7]. The task of stereo matching is exactly this - finding the shift between two image blocks that are assumed to be if not identical, nearly the same. This property has been applied to stereo matching before, both as an assistive method for refining the search area for a more detailed matching algorithm [19] and as the primary matching tool [20, 21]. However, the property is not unique to the Fourier transform. It can be shown that DCT (Discrete Cosine Transform), a special case of DFT (Discrete Fourier Transform) retains this property where the correlation is computed between the signs of the DCT [22]. This property can also be empirically extended to a wide array of different transforms.

The presented transforms (except for census) can be expressed as matrix multiplications for a fixed size transform. An $N \times N$ 2D signal x transformed using

transform T is

$$F = TfT', \tag{2.16}$$

where T' is the transpose of T. However, in practice, an implementation of a transform is rarely done using this method, as there is in many cases much redundancy. The matrix multiplications in Eq. 2.16 are usually unrolled to avoid doing the same computations more than once. Such optimizing scheme is for instance [23].

2.2.1 Transforms

Transforms experimented for applying into sign-only matching are DCT, two integer transforms designed to approximate DCT, Walsh-Hadamard, a simplified version of the Haar wavelet transform and the census transform. In the following, the basics of each candidate transform are described, and the particular matrix presentations used in this work are shown. All of the presented transforms, including census, have the inherent tolerance for intensity differences within the stereo pair caused by different exposure.

Discrete Fourier transform

The DFT (Discrete Fourier Transform) is one of the key techniques in modern digital signal processing. As a discrete version of the continuous Fourier transform, it can be used to express time or spatial domain signals in frequency domain as a combination of sinusoidal waves. The complex valued transform of real valued one-dimensional signal a of size N is defined as

$$A(k) = \sum_{n=0}^{N-1} a(n) e^{-i2\pi \frac{kn}{N}}, k \in [0, N-1]\,. \tag{2.17}$$

DFT can also be presented as a complex valued transform matrix, but as it is not used in this work, the matrix is omitted. For two-dimensional $N \times M$ data, DFT is

$$F(k_1, k_2) = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f(n,m) e^{-i2\pi\left(\frac{nk_1}{N} + \frac{nk_2}{M}\right)}, k_1 \in [0, N-1]\,, k_2 \in [0, M-1]\,. \tag{2.18}$$

An important property is that multiplication in Fourier domain is equivalent to convolution in spatial domain, $FG = x * y$, where $*$ stands for convolution. Among other things, this property leads to the technique Phase-Only Correlation (POC). Cross correlation is essentially computed as a convolution between the spatial signals. When both signals are transformed into the Fourier domain via DFT, the cross correlation can be computed as a multiplication of the transformed signals.

Discrete Cosine transform

Discrete Cosine Transform (DCT) [24] is a tool often applied in various image processing tasks, including compression and denoising. It comes in four different types, of which type II is the most commonly used. Unlike DFT, DCT gives real valued outputs on real valued inputs. One-dimensional type II DCT of size N is defined as

$$X(k) = \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right), k \in [0, N-1] \tag{2.19}$$

In the two-dimensional case, like with DFT, DCT is consecutively applied along the rows of the input data, then along the columns of the intermediary transformed data (or vice versa),

$$X(k1,k2) = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{n1,n2} \cos\left(\frac{\pi}{N_2}\left(n_2+\frac{1}{2}k_2\right)\right)\cos\left(\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right) \tag{2.20}$$

The matrix presentation of an size 8 DCT is

$$T_{DCT} = \begin{bmatrix} 0.354 & 0.354 & 0.354 & 0.354 & 0.354 & 0.354 & 0.354 & 0.354 \\ 0.490 & 0.416 & 0.277 & 0.098 & -0.098 & -0.278 & -0.416 & -0.490 \\ 0.462 & 0.191 & -0.191 & -0.462 & -0.462 & -0.191 & 0.191 & 0.462 \\ 0.416 & -0.098 & -0.490 & -0.278 & 0.278 & 0.490 & 0.098 & -0.416 \\ 0.354 & -0.354 & -0.354 & 0.354 & 0.354 & -0.354 & -0.354 & 0.354 \\ 0.278 & -0.490 & 0.098 & 0.416 & -0.416 & -0.098 & 0.490 & -0.278 \\ 0.191 & -0.462 & 0.462 & -0.191 & -0.191 & 0.462 & -0.462 & 0.191 \\ 0.098 & -0.278 & 0.416 & -0.490 & 0.490 & -0.416 & 0.278 & -0.098 \end{bmatrix}.$$

DCT is an invertible operation, which is often utilized when decompressing images compressed in DCT domain. However, for the purpose of this work, that property is not utilized. Although similarity is computed in the transform domain, the similarity value can be directly mapped to a spatial pixel pair without the need of inverse transforming the blocks or their products.

The applicability of DCT into various purposes has sparked the need for simpler computation of the transform. Transform matrices with integer coefficients have been designed in a way that approximates the original transform and therefore its properties, but can be computed without using floating point arithmetic. Such transforms are for instance [18](later, I-DCT A) and [25](I-DCT B). For example, the transform matrix of I-DCT A is

$$T_{I-DCTA} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & -3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix}.$$

Walsh-Hadamard transform

The Walsh-Hadamard transform can be seen as a binary Fourier representation. Instead of sinusoids, the basis functions of the Walsh-Hadamard transform are Walsh functions, square waves that take two values, either -1 or $+1$. As with the previously described transforms, it can be computed using matrix multiplication. [26] A Walsh-Hadamard matrix is an orthogonal, symmetric transform matrix consisting of values $\in \{-1, +1\}$. As there are no coefficients whose absolute value differs from unity, no multiplication is required when computing the transform, which is an obvious speed-wise benefit. Furthermore, the internal redundancy of the matrix can be exploited by decomposing the computation into steps, where certain operations are cached and used as building blocks of larger composite values.

Sylvester matrices are a type of Walsh-Hadamard matrix, and they can be constructed recursively starting from $H_{2^0} = H_1 = \begin{bmatrix} 1 \end{bmatrix}$ using [27]

$$H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{bmatrix}, \tag{2.21}$$

The transform matrix T_{WH} used when referring to Walsh-Hadamard in this work is $H_{2^3} = H_8$,

$$T_{WH} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}.$$

Haar wavelet transform

Not unlike the Walsh-Hadamard transform, the Haar wavelet transform uses basis functions resembling square waves. However, the basis functions are determined by scaling and translating the simplest possible wavelet function, *Haar wavelet*. [28] The requirement of orthogonality can again be relaxed, therefore avoiding the multiplication with the scalar coefficients of elements in the Haar matrix,

$$T_{Haar} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}.$$

The Haar transform without the scalar coefficients in the individual elements of the matrix is extremely simple to compute. As can be seen from the matrix representation, there is a large number of zeros as coefficients. Therefore when unrolling the matrix multiplication, most of the individual multiplications can simply be dropped. Unrolling one matrix multiplication of T_{Haar} with a 1D signal yields the following 14 operations:

$$\begin{aligned} a_0 &= x_0 + x_1 & X_7 &= x_6 - x_7 \\ a_1 &= x_2 + x_3 & X_6 &= x_4 - x_5 \\ a_2 &= x_4 + x_5 & X_5 &= x_2 - x_7 \\ a_3 &= x_6 + x_7 & X_4 &= x_0 - x_7 \\ b_0 &= a_0 + a_1 & X_3 &= a_0 - a_1 \\ b_1 &= a_2 + a_3 & X_2 &= a_2 - a_3 \\ & & X_1 &= b_0 - b_1 \\ & & X_0 &= b_0 + b_1, \end{aligned} \tag{2.22}$$

where $a_{0..3.}$ and $b_{0..1.}$ are temporary variables storing intermediate results. This allows avoiding having to recompute them for several elements of the output X. In contrast, a naive multiplication with a 8×8 matrix would require 64 multiplications and 56 additions.

Census transform

The census transform differs from the others presented by that it is not applied as a matrix multiplication, but by comparing the pixels inside the window to the

center pixel. This is computationally very simple, as processing of one pixel only takes one memory access and one comparison. The census transform is a common sight in stereo matching algorithms that are ranked high [6], [7] on the Middlebury comparison site [8].

The (m, n) coefficient of a transformed census

$$A(n,m) = \begin{cases} 1, & f(n,m) \geq f(n_0, m_0) \\ 0, & f(n,m) < f(n_0, m_0)0 \end{cases}, \tag{2.23}$$

where (n_0, m_0) is the center point of the spatial domain window.

For the sake of uniformity with the other presented transforms, the census can be expressed as

$$F(n,m) = f(n,m) - f(n_0, m_0). \tag{2.24}$$

This way the same approach of taking the signs of the transform coefficients for compressing the window can be applied to a census transformed window.

2.2.2 Phase-only and sign-only correlation

POC can be used to find the translation between two images, and with proper conversion of the input space (into polar coordinates), also rotation and scaling. [29]. It has often been applied to various image registration purposes [30, 31, 32]. In the particular case of stereo matching, only detecting translations of image segments is required. The link between POC and SOC described in the following is presented in detail in [33].

Consider a 2-dimensional, real valued $N \times M$ image f and its circularly shifted copy g, whose complex valued 2D Discrete Fourier Transforms are F and G. They can both be decomposed into a presentation such as

$$|F(n,m)|F'(n,m) = |F(n,m)| \exp(j\theta_{F(n,m)}), \tag{2.25}$$

where $F' = \exp(j\theta_F)$ is the phase term and $|F|$ the magnitude. To keep the notation clean and readable, the indexing from $F(n,m)$ is omitted in the following, but all the operations are to be considered element-wise. The relative spatial shift is included in the phase terms as the difference between θ_F and θ_G. The cross spectrum R between F and G is given by

$$R = \exp\theta_F|F| \exp -\theta_G|G| = F\overline{G}, \tag{2.26}$$

$\overline{G}$ being the complex conjugate of G. R normalized by the amplitude information is then

$$\hat{R} = \frac{F\overline{G}}{|F\overline{G}|} = F'\overline{G'}. \tag{2.27}$$

The relative estimated translation is given by the *POC function*, $\hat{r}$, which is the inverse transform of $\hat{R}$. The translation between images f and g is the location of the maximum peak in $\hat{r}$. For a perfect match between two identical circularly shifted images, the maximum value of the POC function is 1. For a less than perfect match, e.g. the shift is not circular, or the images are not identical, the corresponding peak value will be between 0 and 1.

Following the same line of reasoning, sign-only correlation (SOC) in DCT domain can be expressed as a special case of POC for a real valued transform,

$$\hat{R} = F'\overline{G'} = F'_{DCT}G'_{DCT}, \tag{2.28}$$

where F'_{DCT} and $G'_{DCT}C$ are the "phase" terms of the DCTs, i.e. the signs of the transform coefficients. In practice they are represented by ones and minus ones. The inverse DCT of this spectral function is again the SOC function,

$$r_{DCT}(p,q) = \frac{1}{NM}\sum_{n=0}^{N}\sum_{m=0}^{M} S_n S_m \hat{R}(n,m) cos\left(\frac{\pi pn}{N}\right) cos\left(\frac{\pi qm}{M}\right), \tag{2.29}$$

where p and q are the displacements evaluated and $\hat{R}(N) = \hat{R}(M) = 0$. By simplifying that $N = M$, S_k can be defined as,

$$S_k = \begin{cases} \frac{1}{2}, & k = 0, N \\ 1, & k = 1, 2, ..., N-1. \end{cases} \tag{2.30}$$

Moreover, as a special case of the sign-only correlation ($n = q = 0$, i.e. 2.29 is evaluated only for no displacement) is the DCT domain similarity between f and g with a $N \times M$ sized transform,

$$s_{DCT}(f,g) = \frac{1}{NM}\sum_{n=1}^{N}\sum_{m=1}^{M} F'_{DCT}(n,m)G'_{DCT}(n,m). \tag{2.31}$$

Eq. 2.31 is then generalized for a generic transform T,

$$s_T(f,g) = \frac{1}{NM}\sum_{n=1}^{N}\sum_{m=1}^{M} F'_T(n,m)G'_T(n,m). \tag{2.32}$$

This metric is proposed as a similarity measure to be used in stereo matching as described in section 2.1.2. The function is evaluated for each pairwise comparison

of blocks from a stereo pair, but some computational optimizations can be done to improve the speed of doing this.

2.2.3 Computing similarity between transformed windows

The information of interest in the suggested transform-based methods consists of either plus or minus signs of the coefficients (which holds also for the formulation of census transform in Eq. 2.24). This information is binary, so it is natural to encode it as such, i.e. as a *bitstring*. The encoded transformed window of n pixels therefore has only the size of n bits, one bit per pixel. This allows for both compact storage of transformed windows, and efficient processing when comparing two windows.

Each transformed window is needed several times as a candidate block when performing the matching in both directions. Therefore it makes sense to store the encoded window after the first transform and to bypass doing it again for each time that same window needs to be evaluated as a matching pair. The storage requirement for all windows centered on a pixel of an $W \times H$ size image is $2nWH$ bits. For instance, for a 1920×1080 Full HD stereo pair and 64 pixels per window, temporarily storing all the transformed windows takes roughly 32 MB of memory, and each windows is transformed only once. Otherwise each disparity estimate would require $d_{min} - d_{max} + 1$ transforms, leading to a huge amount of redundant computation.

The key to the performance of transform-based matching is comparing bitstrings efficiently. The comparison is based on Eq. 2.32. However, a pixel-wise computation would also require several steps iterating through each bitstring, and the $\{0, 1\}$ encoding would have to be decoded back to $\{-1, +1\}$ for the cross-correlation to produce expected results. It is important to note that a XOR-operation between encoded operands $\in \{0, 1\}$ results in the same, but reversed, relative ordering of values of as cross-correlation in $\{-1, +1\}$ with zero displacement (Table 2.1).

Table 2.1: Correlation of signs and their corresponding binary encoding. The symbol $\star$ denotes the cross-correlation between A and B with lag 0,0, i.e. no relative shift between the signals.

A	B	A_{enc}	B_{enc}	$A_{enc} \oplus B_{enc}$	$(A \star B)[0, 0]$
-1	-1	0	0	0	1
-1	$+1$	0	1	1	-1
$+1$	-1	1	0	1	-1
$+1$	$+1$	1	1	0	1

In Eq. 2.32, a single multiplication leads to incrementing the correlation value by one if the operands are similar, or decrementing it by one if they are opposing signs. In XOR, similar operands have no effect on the sum, while opposing signs

increment it by one. It does convert the interpretation of the value from similarity to a cost, but this does not matter, as the only required change is to look for a minimum instead of a maximum. The lack of a penalty term for opposing signs does not matter either, as the relative ordering of the values still stays the same, only scaled to be strictly positive.

It is highly probable that any processor will support logical operations, XOR among them. This allows replacing the sign-wise iteration over the window by a limited number of XOR-operations. In the best case, the XOR part of the correlation can be computed in a single operation on 64bit machines. What remains to be done is collecting the results, i.e. the summation of Eq. 2.31. In practice, this is done by counting the bits that are set to one in the XOR result, i.e. determining the population count.

There are several methods of doing this commonly available in sources on the Internet [34]. In any case, looping through a bitstring and counting one by one whether or not a bit is set is not a reasonable alternative. There are at least three distinctive ways of counting the set bits in a variable. First, there are methods utilizing common commands found in processors, such as logical operations, bitwise shifts and algebraic functionality. These aim to exploit the fact that the data is packed so tightly into single variables that SIMD-like (Single Instruction, Multiple Data) behavior is achieved. Second, if the amount of set bits in an m bit segment is precomputed and arranged in an array according to the numerical value of that segment, it is possible to use the array as a lookup table. When the array is indexed with the segment, the amount of bits set in that segment is the result of that one memory lookup. If the array is kept below the appropriate cache size of the processor, this approach can be very fast. For the n bit bitstring, n/m memory reads have to be performed per bitstring. The division should preferably be even so that there is no wasted memory bandwidth.

The last kind of bit counting is using architecture-specific commands that are designed just for the job. Namely the x86 (and the later 64 bit variants) architecture used in most PC's has an expansion which includes the command POPCNT [35], while the NEON coprocessor found in ARM processors has the VCNT [36] command. Ideally, these kinds of operations can perform the population count on large bitstrings in one clock cycle. Both of these functionalities are aimed at SIMD processing, so this is more than convenient for the matching algorithm, which is embarrassingly parallel at the window comparison level. It is important to note, that the requirement for faster sequential processing is only to have access to a scalar population count operation. It will benefit from the fact that such operations are done in parallel with the same limitations as much as any other algorithm, including the reference method SAD.

3. IMPLEMENTATION

For experimentation with the different transforms in a laboratory setting, a very general implementation relying on Matlab and C++ was made. Individual transforms were not optimized separately, but a library for efficient matrix multiplication was used. For the most frequently used parts, a C++ program was compiled as a Matlab MEX file. The MEX interface allows the programmer to write code on a number of languages, which can run natively on the platform. Through the interface, Matlab can call the program and access its inputs and outputs. Although the interface can be cumbersome at times, porting computationally intensive parts of the code to a representation native to the platform can bring massive improvements in performance. For instance, a transform-based matching algorithm that took nearly 10 minutes in Matlab for a single stereo pair, takes only 0.5 seconds when executed through the MEX interface as a natively compiled C++ program. Needless to say, the flexibility of experimentation is greatly increased.

The use of the MEX interface allowed interfacing between the two parts to benefit from their strengths. Matlab severely lacks performance when working with data consisting of single bits, which was required in compressing and comparing the transformed windows. Therefore Matlab was used mostly for manipulation of the cost volume and analyzing the data, while the transforms and comparisons were accessed via MEX from a natively compiled program.

3.1 Generalized transform-based matching

A straightforward application of the proposed similarity measure into the stereo matching pipeline yields something like Algorithm 3.1. First, Eq. 2.32 is evaluated for each pair-wise comparison of windowed image sections, and the result is stored into a three-dimensional cost volume. The cost volume is then aggregated by filtering it slice by slice. Finally, the minimum cost and its index in the cost volume is found and the disparity estimate is generated. The window for each transform is a rectangular section of the image centered on the pixel (x,y). In the case of a 8×8 window, the window is not actually symmetric, but the center pixel is selected to be one next the center point. As long as the selection is consistent all over the algorithm, this does not have a significant effect.

Algorithm 3.1 The main body of the matching algorithm

```
for x = 1 → width do
    for y = 1 → height do
        f ← WINDOW(L, x, y)                ▷ Window from left image around (x, y)
        F ← TRANSFORM(f)
        for d = d_min → d_max do
            g ← WINDOW(R, x − d, y)        ▷ Window from right image around (x, y)
            G ← TRANSFORM(g)
            C(x, y, d) ← COMPARE(F, G)
for d = d_min → d_max do
    C(:, :, d) ← FILTER(C(:, :, d))        ▷ Filter the xy slice of the structure at d
for x = 1 → width do
    for y = 1 → height do
        for d = d_min → d_max do
            c ← C(x, y, d)
            if c < c_min then
                c_min ← c
                d_cand ← d
        D(x, y) ← d_cand
```

Algorithm 3.2 An optimized version of the matching algorithm which precomputes the transforms. The separate aggregation step is also bypassed on the basis that the windowed transforms already enforces spatial correlation.

```
for x = 1 → width do
    for y = 1 → height do
        f ← WINDOW(L, x, y)                ▷ Window from left image around (x, y)
        L_trans(x, y) ← TRANSFORM(f)
        g ← WINDOW(R, x, y)
        R_trans(x, y) ← TRANSFORM(g)
c_min ← NM
for x = 1 → width do
    for y = 1 → height do
        F ← L_trans(x, y)
        for d = d_min → d_max do
            G ← R_trans(x − d, y)
            c ← COMPARE(F, G)
            if c < c_min then
                c_min ← c
                d_cand ← d
        D(x, y) ← d_cand
```

There is, however, a huge redundancy in recomputing the transforms at each evaluation of the similarity measure. By sacrificing some memory space to accommodate the transformed windows, significant reduction in computation is gained. Also, for inputs where the fixed size of the transform behind the similarity measure is large enough in relation to the image size, the aggregation step may not be necessary at all. After these two modifications, the matching is formulated as Algorithm 3.2.

The most time consuming tasks in the algorithm are the TRANSFORM and COMPARE functions. Their efficient implementation is strongly dependent on the hardware and software environment, and in the case of TRANSFORM, the actual transform that is selected. In some cases, especially if the transform is DCT, there may even be a possibility to use hardware acceleration to compute the transforms.

The lookup table is a relatively hardware-independent solution for comparing transformed windows. The way to do it is, however, very specific to the programming language. Therefore generating a 16 bit lookup table in C++ is presented in Algorithm 3.3, and the usage of the table in Algorithm 3.4.

Algorithm 3.3 Generating a 16 bit lookup table in C++ for performing the population count

```
const int bitsLen = 16;
unsigned countLen= pow(2.0, bitsLen);
unsigned count;
uint8* bitcountArray = new uint8[countLen];

// Count the set bits in all permutations achievable with bitsLen
for ( unsigned i = 0; i < countLen; i++)
{
   v = i;
   // Shift the value left until it evaluates as zero
   for ( count = 0; v; v >>=1)
   {
      // Increment counter if the LSB of v is 1
      count += v & 1;
   }
   bitcountArray[i] = count;
}
```

With 16 bits, there are 2^{16} possible values. The generator iterates through them all, and counts the number of set bits for each. It uses a naive counting method, but as this table needs only to be generated once at the start of the application, this is not a performance concern. It could also be precomputed at compile time and stored with the executable. The table is constructed to allow indexing into it with a variable to return the variables population count. Therefore the population count of each of the bit permutations Algorithm 3.3 goes through is stored into the index pointed by the integer interpretation of the bit permutation itself.

When computing the similarity of two 64 bit presentations of transformed windows, the first step is the *exclusive or* (operator "^"). The value is partitioned into appropriate sized segments, which in this case is a *short int*, a 16 bit (again, in this environment) variable corresponding to the size of the lookup table generated earlier. The partition is efficiently done by interpreting the pointer to the original 64 bit variable as a 16 bit pointer using *reinterpret_cast*. Dereferencing the pointer will return 16 bits from the start of the value. Incrementing the pointer will iterate through the 64 bit variable in 16 bit segments. Indexing the lookup table with the 16 bit segment will return the population count of that segment. After $64/16 = 4$ iterations, the sum will contain the population count of the whole variable.

Algorithm 3.4 A C++ implementation for comparing two transformed windows using the lookup table created in Algorithm 3.3

```
int compare( long long int B1, long long int B2)
{
   int sum = 0;
   long long int cmp;
   cmp = B1^B2;
   unsigned short * ptr;
   ptr = reinterpret_cast<short int*>(&cmp);

   for ( int i = 0; i < sizeof(long long int)/sizeof(short int); i++)
   {
      sum += bitcountArray[*ptr];
      ptr++;
   }
return sum;
```

3.2 Census-based matching for mobile

The algorithm described was implemented on a prototype mobile device based on the Nokia N950. The result is a C++ class without any external dependencies other than the NEON headers (which are commonly available for the GCC compiler). This provides as much flexibility as possible, as it does not constrain it to any specific operating system or environment. The sections implemented in NEON intrinsics will not compile on other architectures. However, there are also generic versions of the functionality of those sections available interleaved in the program code, accessible by defining a certain compiler macro.

The implemented parts are census transform-based matching and post processing via a histogram-based median filter and a decomposed bilateral filter. Considering the computational aspect, the bilateral filter is most expensive of the discussed post processing methods. However, after the implementation was completed and after no longer having access to the platform, it was noted that improved results can

be achieved by replacing the bilateral filter with a consistency check. Left-right consistency check is ideally only a few operations and two memory accesses per pixel. The work done by the median filter-based occlusion filling is a subset of a median filter that goes through the whole image. Therefore the combination is both faster and offers better quality. The left-right consistency check does not provide usable results if it is done on the raw disparity maps generated by census matching. Therefore it is important to first run median filtering on both maps to remove most of the noise.

3.2.1 Implementation platform

OMAP is a line of system on a chip -products by Texas Instruments. It includes a number of key components for a functional computer in a compact package, and is therefore well suited for mobile applications. OMAP devices come equipped with ARM processors and may offer varying support for SIMD computing in the form of a NEON coprocessor and a DSP. The OMAP 3 the device used in this work is based on usually runs a single core processor in the 500-1000MHz range, while the this particular device is clocked at 1000MHz. [37]

ARM processors are frequently accompanied with a SIMD extension, a NEON coprocessor. The NEON core can execute a large amount of different operations that process several variables at once. The SIMD width of NEON is 128 bits, although some implementations only execute half of this truly in parallel. NEON has its own data types, which are in some use cases completely hidden from the programmer, while more deeply integrated development requires defining and handling these. Those data types consist of vectorized versions of the normal types; different sized signed and unsigned integers, floating point numbers etc. The length of the vectors is limited by the 128bit processing pipeline, i.e. it can fit up to 16 8-bit integers, or 4 of 32-bit floating point presentations etc. [36]

3.2.2 Optimization strategies for NEON

There are three possible ways of utilizing the SIMD properties of the NEON core when writing code targeting an ARM processor. The simplest one is to enable the compiler flags for automatic vectorization and NEON usage. The compiler will then analyze the code and automatically generate machine code that uses the NEON instruction set where it thinks it is beneficial. To aid the process, the programmer can write pragmas inside the code to give additional information about things like the number of iterations certain loops will always perform, or the dependencies of function pointers between iterations.

Another method is to use intrinsic functions, specifically defined C-style functions

that in theory map directly to individual NEON instructions. The programmer is given almost assembly-like control over the data processing, but with a somewhat more intuitive interface. Each operation supported by the NEON core has its own intrinsic function, which is characterized by the operation itself and also by the data types of the parameters and the output.

Finally, the NEON operations can be accessed with raw assembly commands. By manually writing the assembly code, the maximum potential of the NEON core can be reached. However, as with all direct assembly development, this approach is very slow and complicated, especially for debugging. The difficulties of writing assembly suggests that resorting to this level of optimization should be thoroughly considered and is probably best used only in the last stages of creating an actual product.

The compiler oriented optimization strategies do have their own drawbacks. The compiler's capability of discovering the parallel potential in the source code, even with additional information provided by the programmer, is limited. There are several automatic checks done to confirm that those code segments considered to be parallelized will not cause runtime errors or make the code produce erroneous results. The most significant aspect to check is that operations within iterations of the same loop do not access the same data via pointers. These checks appear to remain on the safe side, which is understandable. While in certain applications compromises between accuracy of results and processing speed may be desirable, the errors caused by concurrency and their scale can not be reliably predicted. If automatic parallelization is desired, the programmer must take care that these tests pass.

When using intrinsic functions, parallelization is explicitly defined by the programmer. The compiler then only has to map those functions into the corresponding NEON operations, at least in theory. There seems to be some problems in the mapping stage, though. Conceptually, utilizing the NEON core can boost the performance of simple algorithms 4- or even 8-fold, depending on the version of the hardware [36]. However, this is assuming the operations are used optimally. It is apparently difficult for the compiler to arrange the data operations between the registers of the ARM processor and dedicated NEON registers. This leads to a large amount of unnecessary data transfers between those two, which take significant amount of clock cycles and therefore eat away much of the potential gain from using NEON. To avoid this, manual cleaning of the compiler generated assembly is required, making it arduous to experiment with different parallelization approaches.

For this implementation, a hybrid approach between the first and the second methods was chosen. The most computationally intensive sections of the matching, namely transforming windows and comparing bitstrings were implemented using intrinsic functions. The compiler's attempts at this proved to be inefficient, as

despite of efforts to try to formulate the code in the necessary way, it was not able to do any automatic parallelization. The post processing consists of simpler tasks, so it was left for the compiler to autovectorize, which it performed with varying success.

3.2.3 Implementation specifics

Despite the poor quality census-based matching showed in the preliminary tests, the approach that was finally chosen for the mobile implementation is to completely avoid the aggregation phase and to attempt to compensate with suitable post processing. The matching algorithm is exactly the same as described in section 3.2. The difference is that census is used instead of the heavier decomposing transforms.

The C++ version of the census transform is described for reference in Algorithm 3.5. The NEON accelerated transform is given in Algorithm 3.6. Both implementations of it are functionally equivalent. The custom type bitstr is defined to be long long int for the remainder of the presented code.

Algorithm 3.5 A serial C++ implementation for applying census transform on a window centered at (x, y) in image *img*

```
typedef long long int bitstr
bitstr census( const uint8* img, unsigned x, unsigned y )
{
   uint8 t, c = img[ y * WIDTH + x];
   bool cmp;
   int i = 0;
   bitstr ans = 0;

   for ( int yi = -3; yi <= 4; yi++)
   {
      for ( int xi = -3; xi <= 4; xi++, i++)
      {
         t = img[ (y+yi) * WIDTH + x+xi];
         cmp = t > c;
         ans |= cmp << i;
      }
   }
   return ans;
}
```

The C++ version simply takes the center pixel and loops through the 8×8 window one pixel at a time. At each position, the target pixel t is compared to the center pixel c and the result stored in the boolean variable *cmp*. Shifting the boolean variable by i causes an implicit typecast into an integer, which has the ith bit set to the comparison result. The result bit is then saved at the correct position in the *result* bitstr, which is finally returned to the caller after the whole window has been processed.

Algorithm 3.6 A serial C++ implementation for applying census transform on a window using NEON intrinsic functions

```
bitstr censusmatch::census( const uint8* img, unsigned x, unsigned y )
{
     // Center pixel value duplicated onto 8 lanes
      uint8x8_t cdup = vld1_dup_u8(img+y*width_+x);
      // Bit mask that is used to copy individual bits to the result
      uint8x8_t mask = vdup_n_u8(1);

      uint8x8_t trow;
      uint8x8_t cmprow;
      uint8x8_t result = {0};

      for ( int yi = -3; yi <= 4; yi++)
      {
         // Load a horizontal line of 8 pixels from the window
         trow = vld1_u8( img + (y+yi)*width_ + x );
         // Compare them to the center pixel
         cmprow = vcge_u8(trow, cdup);
         // Bits chosen by 'mask' from 'cmprow' go into 'result'
         result = vbsl_u8(mask, cmprow, result);
         // Shift to pick the next bits on next iteration
         mask = vshl_n_u8(mask, 1);
      }
      ans = reinterpret_cast<bitstr>(result);
}
```

In the NEON version of the transform, the center pixel is also fetched, but the result is duplicated onto the eight lanes of the uint8x8_t NEON variable *cdup*. On each iteration of the for-loop, eight variables at a time are loaded from memory to *trow* in a single command. The whole row is compared to the duplicated center pixel *cdup*. The inner loop of Algorithm 3.5 is therefore replaced by the parallel processing of a complete row of the window in one iteration. The result of the comparison is stored in an uint8x8_t variable, where all bits of a lane are set to either 0 or 1 based on the relative values stored in the corresponding lanes of the operands. A single bit is copied from each lane to the result variable. As all the source bits of a lane are the same, it does not matter which one is copied. This allows the copy operation to directly copy a bit from the same location that is the correct location in the result. This is done using a binary mask that has only one bit selected per lane. The mask is shifted by one on each iteration, moving the destination of the result bit in the *result* variable. Finally, the variable of type uint8x8_t is simply reinterpreted as a bitstr (i.e. long long int). As the variables are of the same size (64 bits vs. 8×8 bits), this makes the function compatible with the rest of the program flow without porting everything over to NEON data types.

Algorithm 3.7 A C++ implementation for comparing two transformed windows using NEON intrinsic functions

```
for ( int y = 0; y < HEIGHT; y++ )
{
   for ( int x = 0; x < WIDTH; x++ )
   {
      // Read and duplicate the center pixels of four windows
      uint16* sptr = reinterpret_cast<uint16>(fromstrs);
      uint16x4_t B10 = vld1_dup_u16(sptr + 4*(y*WIDTH + x));
      uint16x4_t B11 = vld1_dup_u16(sptr + 4*(y*WIDTH + x) + 1);
      uint16x4_t B12 = vld1_dup_u16(sptr + 4*(y*WIDTH + x) + 2);
      uint16x4_t B13 = vld1_dup_u16(sptr + 4*(y*WIDTH + x) + 3);

      bitstr*  toptr = tostrs + width_*y + x;
      for ( int d = mindisp_; d <= maxdisp_; d+=4)
      {
         xd = x + d;
         uint16x4_t sums = {0, 0, 0, 0};

         // Read four complete pixels and distribute them
         uint16x4x4_t B2 = vld4_u16(reinterpret_cast<uint16*>(toptr + d));
         uint16x4_t B20 = B2.val[0];
         uint16x4_t B21 = B2.val[1];
         uint16x4_t B22 = B2.val[2];
         uint16x4_t B23 = B2.val[3];

         // XOR, count bits and accumulate sum x4
         uint16x4_t cmp = veor_u16 (B10, B20);
         uint8x8_t counts = vcnt_u8( vreinterpret_u8_u16(cmp));
         sums = vpadal_u8( sums, counts);

         cmp = veor_u16 (B11, B21);
         counts = vcnt_u8( vreinterpret_u8_u16(cmp));
         sums = vpadal_u8( sums, counts);

         cmp = veor_u16 (B12, B22);
         counts = vcnt_u8( vreinterpret_u8_u16(cmp));
         sums = vpadal_u8( sums, counts);

         cmp = veor_u16 (B13, B23);
         counts = vcnt_u8( vreinterpret_u8_u16(cmp));
         sums = vpadal_u8( sums, counts);

         // Store all 4 computed similarities
         vst1_u16(costvalues+d, sums);
      }
      d = find_min(costvalues);
   }
}
```

The other focal point of the algorithm is the comparisons of transformed windows. The plain C++ version of the mobile implementation is roughly the same as described in conjunction with the generalized version in Algorithm 3.3 and 3.4. The comparison function is called once for each of the pairwise comparisons done between transformed windows. The NEON version is somewhat more complex, as several comparisons are parallelized to be performed simultaneously. The straightforward method would be to fill a single NEON register with one transformed window. This is however avoided due to the fact that it is more efficient to aggregate the results along individual NEON lanes instead of across a single NEON variable. The memory accesses resulting from traversing the search range using $x \pm d$ leads to over indexing at certain areas. Handling this is trivial by assigning an artificially high cost to those comparisons, and is omitted from the program code.

For each coordinate (x, y) in the source image, the corresponding compressed, transformed window is retrieved. The pointer to the window storage is interpreted as a pointer to 16 bit values - incrementing and dereferencing this pointer will return 16 bit segments, which are duplicated to fit a variable of type uint16x4. After this, the variables *B10...B13* each contain a duplicated 16 bit segment of the reference window. The disparity search range is then iterated in steps of four, with each iteration comparing four target windows with the reference window.

A wide memory load is used to read four compressed target windows into a composite variable. The variable is then decomposed into separate vectors so that each one contains a 16 bit segment of each of the target windows. For instance, *B20* will contain the first 16 bits from all of the target windows. Now the arrangement of the data between the reference (*B10...B13*) and the targets (*B20...B23*) is consistent. This allows exclusive or and population count to be performed in one operation each on corresponding segments of the windows to be compared. The population count operation takes operands as 8 bit variables, but fortunately 8 at a time. As the operation is not dependent on the actual numerical interpretation of the operands and the existing uint16x4_t variables are of the same length, a simple reinterpret_cast will suffice.

This is repeated for each segment while aggregating the resulting sum into *sums* after each population count. Therefore it is not necessary to cross the lane borders to sum the results, which would not be easily achievable. The final result is stored in a vectorized variable where each element contains the value for one pairwise comparison. They can be directly written to memory using a wide store operation. After the whole search range has been processed at coordinates (x, y) , the minimum value is found like described in Algorithm 3.2.

4. RESULTS AND EVALUATION

There are a number of quality metrics that appear frequently in the context of evaluating quality of depth maps. The percentage of bad pixels (PBP) is perhaps the most common. Among other things, it is the primary metric used on the Middlebury evaluation site, which ranks submitted stereo matching algorithms by the quality they produce on a common test set. The disparity map is compared to the ground truth, and those pixels that are different are counted. PBP is then the percentage of those differing pixels from the whole image. It is reasonable to allow a slight difference, e.g. a variation of one. The proposed similarity measure is not compared to those reported on the site, as it would not be a fair comparison. The proposed approach only covers a part of the stereo matching pipeline, and is not a complete matching system like those ranked on the site. [8] PBP is still used as the metric to compare how variations in the proposed preliminary stages of the matching pipeline affect the quality. Another metric with which results are occasionally presented is mean squared error (MSE) between the disparity map and the ground truth.

Due to the somewhat erratic nature of the results given by census matching and the specific use case of virtual view rendering is aimed at, an additional metric is also considered when dealing with the the mobile implementation. PSNR is the peak-signal-to-noise ratio computed between the disparity estimate and the ground truth. PSNR_R is computed between novel views rendered based on the estimated disparity map and the ground truth. The absolute quality is obviously dependent on the used rendering method, but as all images are generated with the same method, the relative difference does indicate how the disparity maps rank in relation to each other. [24]

The problem with numerical metrics is that they tend to measure reconstruction accuracy, which is not necessary equivalent with the quality of rendering. Some metrics are strongly affected by types of errors that do not have much of an impact on the perceived quality of rendered views, and vice versa. Even though metrics like PSNR_R attempt to evaluate the perceived quality of a viewer, it is still affected by artifacts that may be negligible to the viewer. For instance, a error of one in the disparity map can cause a large portion of the scene to be shifted by one pixel. A viewer will not even notice such a difference, but PSNR can be severely affected if the area has high frequency texture. Therefore a small scale subjective test was

performed to compare the quality of scenes rendered from estimates by SAD and census matching. Subjective testing is described in detail in section 4.2.2.

4.1 Generalized transform-based methods

The data set for these experiments is 21 pre-rectified Middlebury stereo pairs, which comes in difference sizes. The majority of the testing is done using images scaled to one third of the original size, while some experiments are conducted on the full size images (approximately 1300×1100 pixels). [38]. Occluded areas are extracted from the ground truths and are excluded from the numerical quality estimates, as any kind of occlusion filling is not being handled or proposed. For SAD, the radius of the box filter used in cost volume aggregation for each stereo pair is always selected as the one providing best results in terms of that specific metric. Quality produced by SAD is greatly dependent on the size of the aggregation window, so this approach gives even a slightly too favorable treatment to the method. In practice, the window size would be either constant or adaptive based on some features of the images. Selecting the optimal as done here is not possible in a real application. The metrics have to be computed in reference to a ground truth, which is obviously not available if one must resort to stereo matching to begin with.

The reference point used for comparing the results of transform-based metrics is the commonly used sum of absolute differences (SAD) with different sizes of rectangular aggregation windows. The quality of disparity estimates is evaluated with the percentage of pixels that do not match the ground truth (PBP) and the MSE of the estimate against the ground truth. As shown in Table 4.1, all the tested transforms perform on similar levels. This also holds true for the majority of individual comparisons. Therefore the rest of the results are presented only for DCT and Haar in order to improve the clarity of the results. DCT is selected due to its widespread usage in image processing tasks and Haar as the one providing the best results and offering the fastest computation. It is worth noting that there are differences in the quality of the two integer transforms designed to appriximate DCT. I-DCT A performs slightly worse than true DCT, while I-DCT B is actually better. The scaling of the algorithm is exactly the same as described in section 4.2.1 regardless of the transform, only with a higher constant component before any comparisons are made.

Table 4.1: Average quality metrics over the whole dataset for each of the transforms experimented with.

	SAD	DCT	I-DCT A	I-DCT B	Walsh-Had.	Haar
PBP	34.0%	28.3%	29.3%	27.8%	27.0%	26.2%
MSE	148.6	149.2	155.7	148.0	142.6	142.2

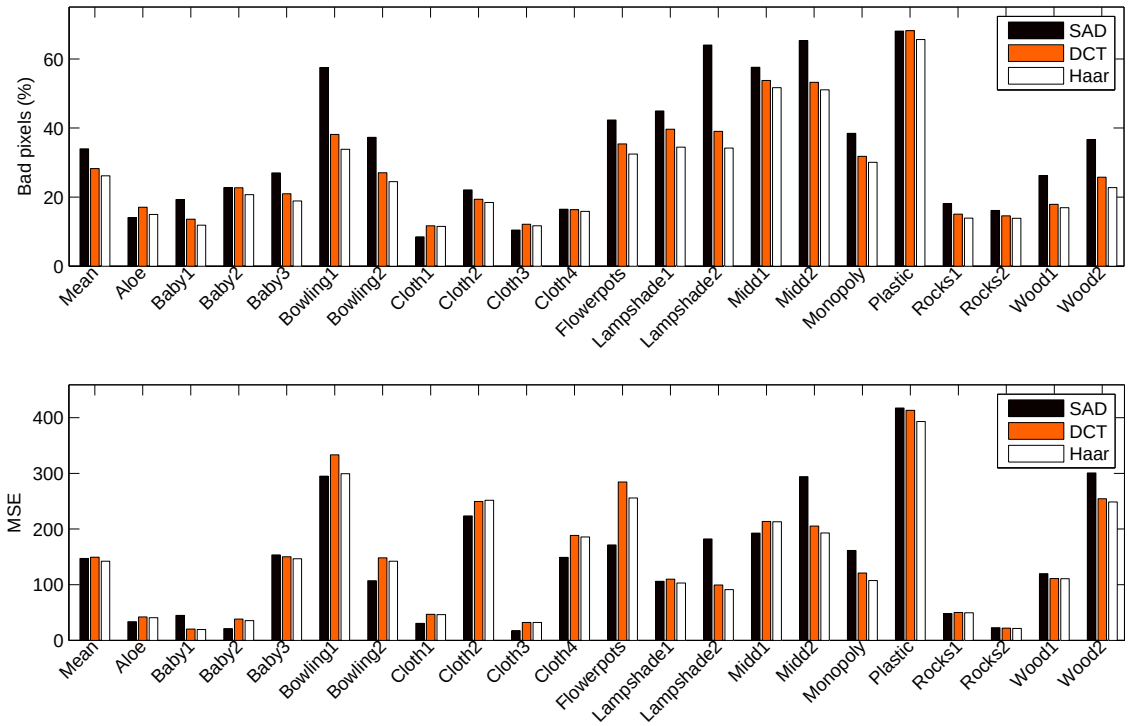


Figure 4.1: Percentage of bad pixels and MSE of estimated disparity map for each image in the Middlebury 2006 data set using SAD, DCT and Haar -based similarity measures. All disparity maps have been 5×5 median filtered

Figure 4.1 shows percentage of bad pixels and MSE for each individual stereo pair in the data set. The cost volume formed by transform-based methods has not been aggregated in any way. The measurements for the SAD method are using the best aggregation window size from the tested range (from 1×1 to 19×19). The only post processing that has been applied after the matching stage is median filtering. The general trend is that transform-based methods give disparity estimates with less bad pixels for the majority of the pairs, and do not fall very far behind even in the cases where SAD performs better. In terms of MSE there is some more variation between stereo pairs, with the averages over the whole data set being roughly equal. This leads to the conclusion that transform-based methods make bigger mistakes, but are wrong less often than SAD.

In order to improve the quality of the disparity estimate, the cost volume of transform-based methods can also be aggregated. Figure 4.2 demonstrates the effect of a box aggregation step on the cost volume generated from images of the smaller data set. SAD does not in practice work without the cost aggregation step, so 1x1 (no aggregation) has been omitted for that method. Both DCT and Haar -based methods are getting some additional benefit from aggregation up until the window sizes 9×9 and 11×11. In terms of MSE, the cutoff of quality improvement at 9×9 is very pronounced. This coincides somewhat with the window size of the transform,

which is 8×8, although the reasoning why they seem to be linked is not entirely clear.

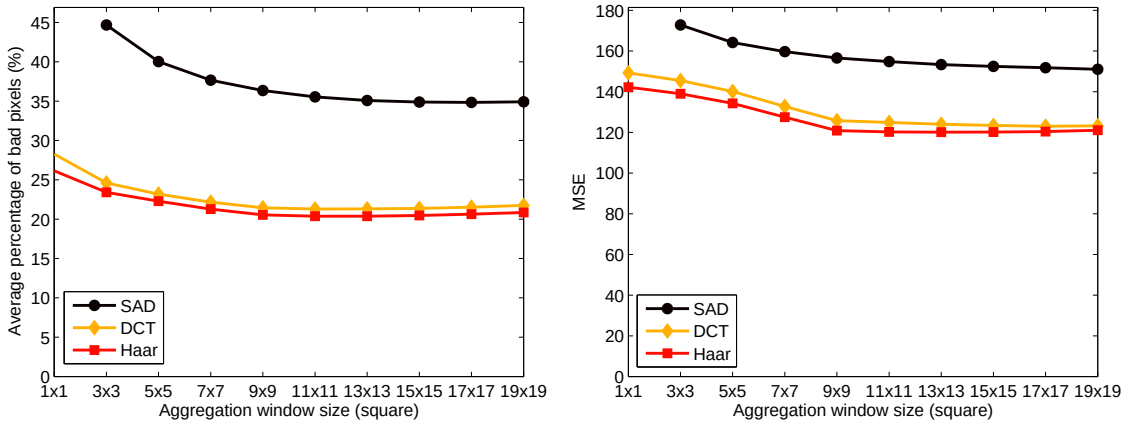


Figure 4.2: Average percentage of bad pixels and MSE for the Middlebury 2006 1/3 data set after box filtering (i.e. aggregating) the cost volume with different block sizes. Size 1x1 corresponds to no filtering, which is omitted for SAD as it simply does not work.

For full size images, Figure 4.3 shows that the benefit gained from aggregating the transform-based cost volume is larger than on the smaller images. This is due to the fixed size of the transform window, which represents smaller sections of the image as the image size increases. Here, SAD matching with appropriate sized aggregation window (in the range of 21-31 pixels) actually reaches the same level of quality as the methods based on DCT and Haar without aggregation. By sacrificing the speed gain over SAD by incorporating the cost aggregation step, those will again surpass it in quality, reaching the minimum PBP around the same window sizes as SAD.

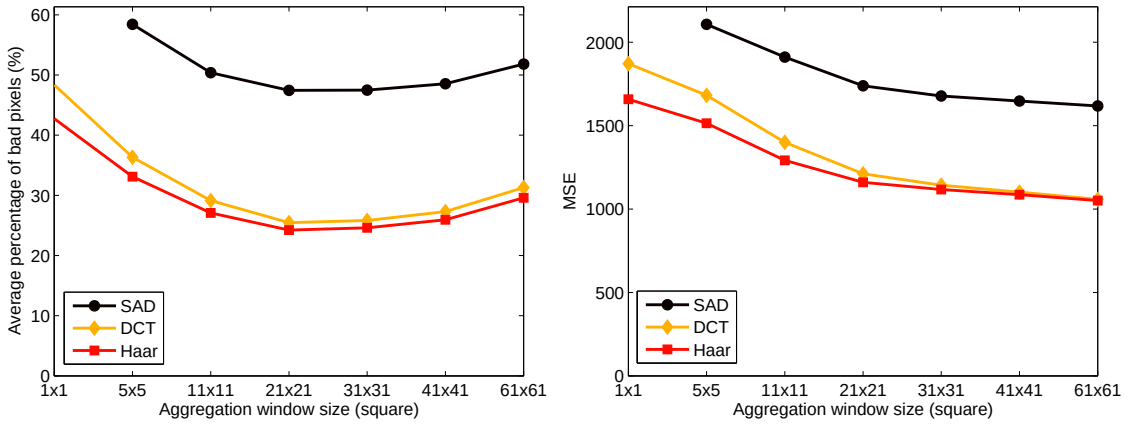


Figure 4.3: Average percentage of bad pixels and MSE for the Middlebury 2006 full size data set after box filtering the cost volume with different block sizes. In contrast to the smaller resolution data set, there is a slight overlap in the average quality between SAD and the others.

As mentioned earlier in subsection 2.1.5, based on the type of noise introduced into the disparity map by the typical mistakes transform-based methods do, median filtering can be recommended as a post processing step following immediately after the disparity computation. Figure 4.4 demonstrates the effect of median filtering on the average PBP and MSE of disparity maps computed from the smaller data set. As is evident, even a moderate amount of filtering significantly improves the quality, confirming the hypothesis. It is also clear, that the type of noise in SAD matched disparity maps is different, as the median filter has very little effect on it. PBP for transform-based methods improves around 5-6 percentage units, while the effect for SAD less than 2 percentage units. The optimal filter window in this case is 9×9, which is about the same as the optimal aggregation window, i.e. slightly more than the original matching window of 8×8. If filtering is applied with too large of a window, the details in the edges of the shapes start disappearing.

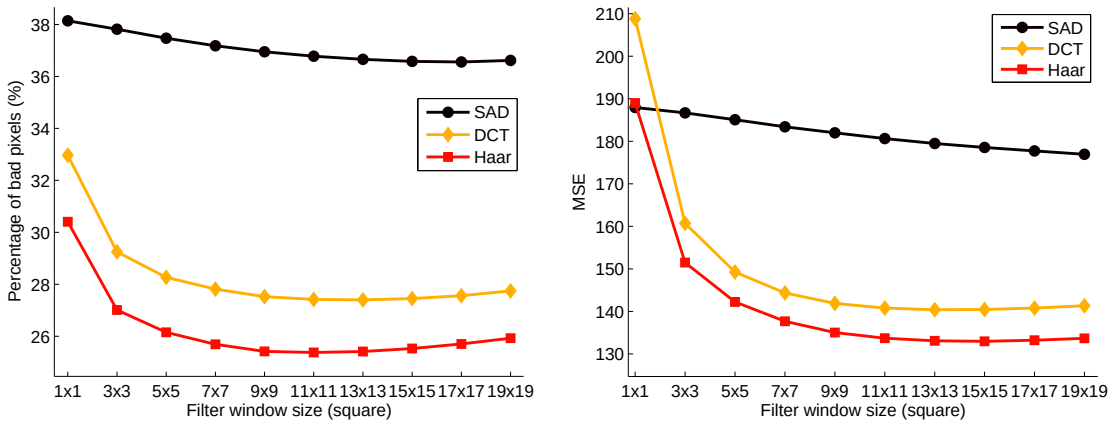


Figure 4.4: The effect of median filtering the disparity map with different sized windows on the quality of the disparity estimate. While 5x5 SAD is not the optimal window size for cost volume aggregation for all images, it is used to display the trend for SAD. Transform methods are again not separately aggregated.

The effect of aggregation and median filtering is compared in Figure 4.5. The difference in favor of aggregation is approximately only 2.5 percentage units in the smaller size data set. Forgoing the speed considerations and applying both filtering approaches will improve the quality even more. The visual effect of aggregation is demonstrated in the examples in Appendix 2. With larger images (full size data set), the lesser influence of median filtering the disparity map in comparison to aggregation is shown in Figure 4.6, where the aggregation drastically drops the error rate, while median filtering has only a relatively minor effect.

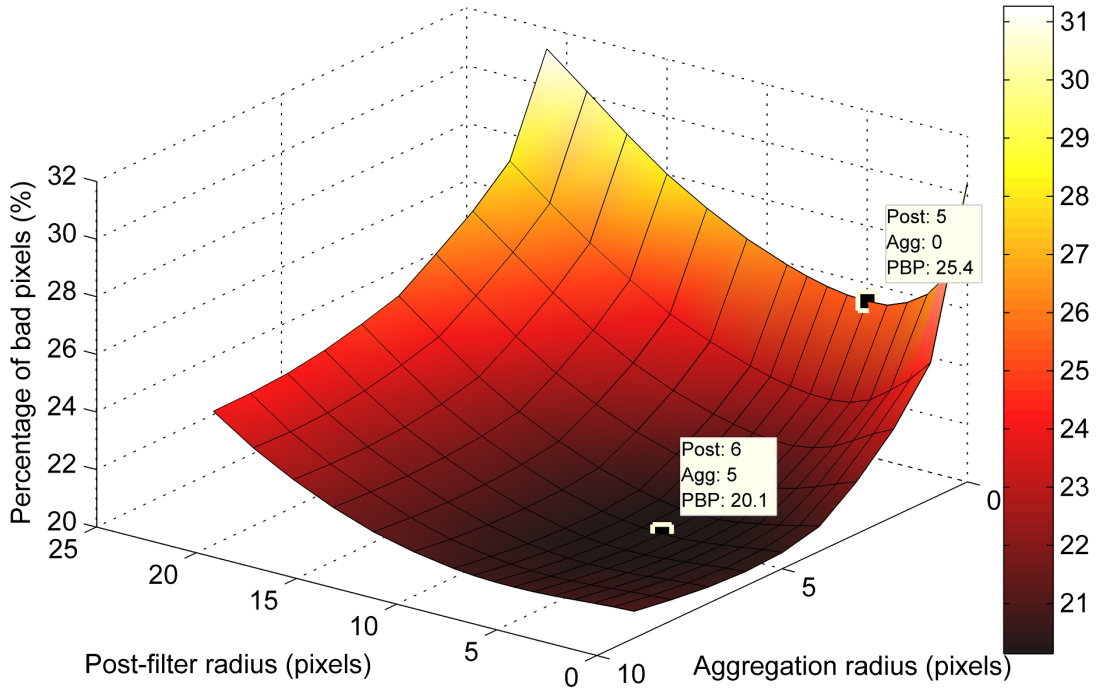


Figure 4.5: The effect of cost volume aggregation and post processing with a median filter on disparity maps estimated from the smaller data set.

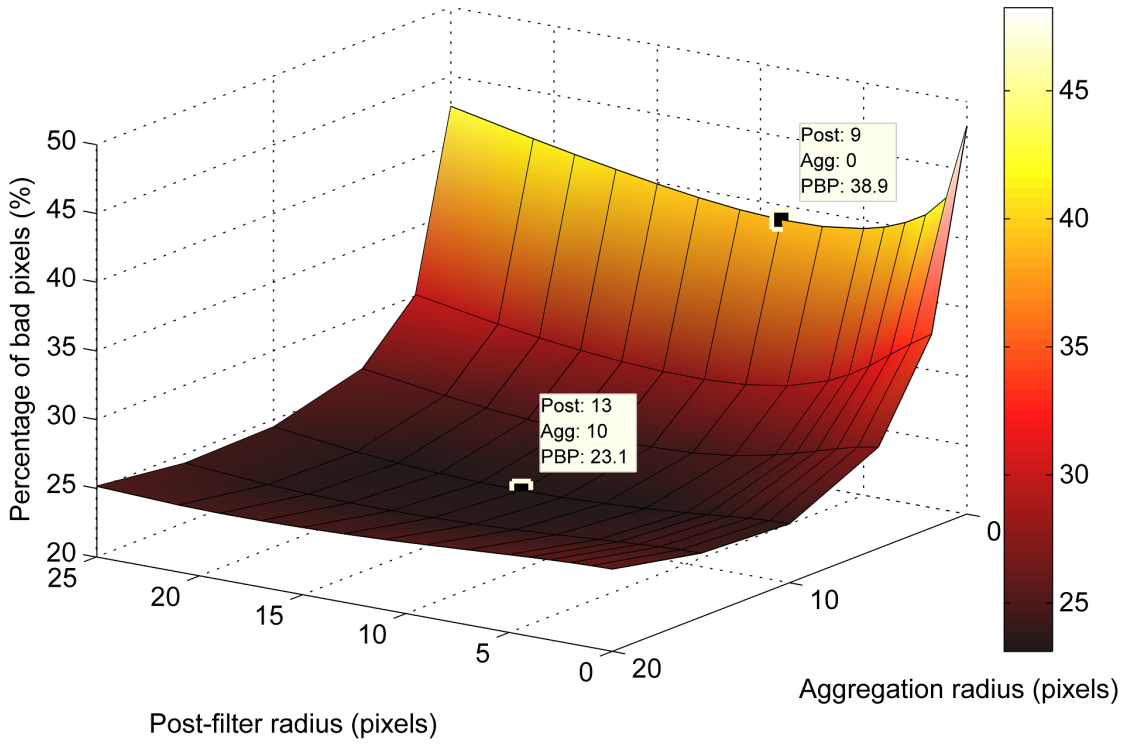


Figure 4.6: The effect of cost volume aggregation and post processing with a median filter on disparity maps estimated from full size data set.

Two examples of disparity maps estimated with SAD, DCT and Haar are shown in Figure 4.7. Worth noting is the difference in the structure of the erroneous matches. SAD experiences large, smooth patches of wrong matches. This can mostly be attrubuted to the the aggregation procedure, which is responsible for much of the outcome. If the errors are dominant in some area, then instead of improving quality, the aggregation step spreads the error around. On the other hand, transform-based approaches suffer from more varying valued errors in matches, in a sense distributing the faulty matches more evenly over the image. This is beneficial as there is then some correct information available at most parts of the scene. There is very little difference in the behavior of the methods in textured surfaces which exhibit piecewise smooth disparities.

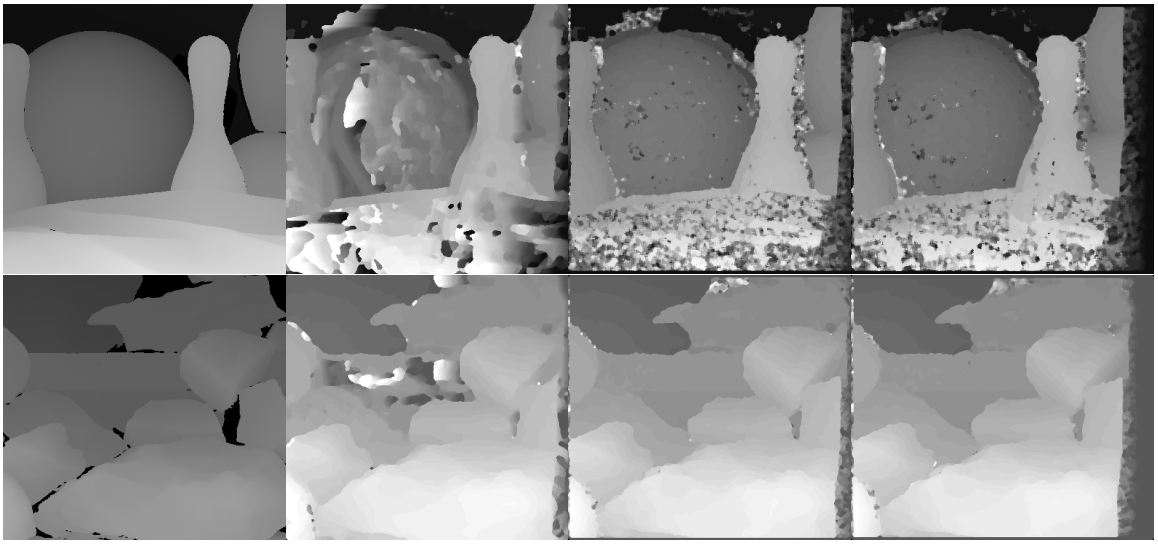


Figure 4.7: Disparity maps of Bowling1(upper row) and Rocks1 (lower), from left to right: Ground truth, estimates with SAD with box filter, DCT and Haar. All maps have been 5x5 median filtered. The Bowling1 scene illustrates well the types of "noise" characteristically originating from the methods. SAD suffers from large smooth patches of wrong disparities while transform-based methods experience a kind of salt&pepper type of noise. The Rocks1 scene shows how all methods behave similarly on smooth surfaces

Middlebury also offers versions of the basic data set which have been taken in different lighting conditions and with different exposure times [38]. Table 4.2 shows a number of tests where the images of the stereo pair have been selected from different exposures. Under each different illumination (Illum 1..3) each different exposure has been compared with the longest exposure time and the pair is marked in the Exp-column like $2, 0$, i.e. exposure number two is paired up with number zero. The indexing corresponds to the one used in the Middlebury data set. SAD has very little success in dealing with any discrepancies in the exposure time, and only provides reasonable results when paired up with the same exposure (that is, $2, 2$). The transform-based methods are affected, but to a much lesser extent. Even

the worst case quality difference between the same exposures and the maximally different exposures (DCT, illum. 1, 2,0 vs. 2,2) is less than 10 percentage units. The aggregated versions of DCT and Haar are especially robust in this respect.

Table 4.2: Average percentage of bad pixels of disparity estimates between images captured with varying exposure times under different illuminations. A-DCT and A-Haar have the optional box aggregation step included. An example of the contents of the data set is presented in Appendix 1.

	Exp	**SAD**	**DCT**	**Haar**	**A-DCT**	**A-Haar**
Illum. 1	2,0	93.0	36.2	33.9	24.5	23.8
	2,1	93.0	28.4	26.3	20.7	19.8
	2,2	35.0	26.9	25.0	20.1	19.4
Illum. 2	2,0	93.2	34.5	32.4	23.4	22.9
	2,1	92.7	29.2	27.5	21.4	21.1
	2,2	35.2	29.4	27.8	21.7	21.4
Illum. 3	2,0	93.2	38.9	37.2	27.0	26.9
	2,1	92.7	32.0	30.5	24.5	24.2
	2,2	43.1	30.9	29.7	24.0	23.9

4.2 Census-based mobile implementation

Table 4.3 displays some numerical quality metrics comparing census and SAD. A slightly different test setting was used than with the generalized transform-based methods, which makes these values only comparable to each other and not with the ones presented earlier. For instance, occluded pixels have not been excluded from this comparison.

Table 4.3: Objective quality metrics comparing disparity maps computed using SAD and census

	PBP (%)	PSNR (dB)	PSNR_R (dB)
Box SAD 11x11	36.8	16.9	31.2
Box SAD 7x7	38.5	16.8	31.0
Census 8x8	46.6	17.4	27.9

4.2.1 Performance analysis

The processing speed was measured on the implementation platform for grayscale images with 433x370 resolution and 70 disparity estimates. Some inherent properties of the platform reduced the timing accuracy to a certain degree, but the results were deemed sufficient to give a sense of the computational load. On a PC, the implementation without explicit parallelization on thread or instruction level, the

implementation took approximately 0.5 seconds (using lookup table for population count). The direct port of that on the OMAP platform had a runtime of 6.7 seconds per pair. Optimizing the memory access patterns dropped this down to 4.2 seconds per pair, and adding the post processing brought it back up to 6.7 seconds.

Implementing transforms and matching with NEON intrinsic functions and enabling the compiler to autovectorize the post processing resulted in a runtime of 4.3 seconds per pair. The matching step alone takes about 2.5 seconds for disparity estimates from both viewpoints of the stereo pair. It can clearly be seen the improvement gained from the NEON intrinsics is nowhere near the maximum potential of the NEON core, which leads to believe that the conversion from intrinsic functions to assembly is not working as well as it should.

A sparse disparity estimate was also tested, where transforms and matching is only computed for every other pixel. The result is then upsampled (by a factor of 2) to the original size using the bilateral filter. This has the advantage that computationally it does not matter if the input of the bilateral filter is a subsampled version. The only difference is with indexing, where indices into the image must be scaled to match the subsampled image size. As in this case the factor is two, this can be done trivially by shifting the index towards the least significant bit. With some loss of quality, the speed of that approach is 2.0 seconds per pair.

The other suggestion for post processing is using a left-right consistency check, which was not implemented due to scheduling constraints concerning access to the device. It is, however, drastically faster than the bilateral filter. Judging by the complexity of the computation, the consistency check is almost trivial to compute in comparison to the bilateral filter. Filling the inconsistent pixels has to be done though, which e.g. corresponds roughly to a second application of the median filter. In any case, it is a much faster approach than the bilateral filter.

Table 4.4: Estimated operation counts required to perform one comparison of windows. Listed methods are box filtered SAD (with summed area tables) and two approaches to comparing transform-based bit strings (with 16 bit memory lookup or hardware implemented population count). Memory cache also plays an important role, but is more difficult to analyze.

	Arithmetic op	Memory reads	Memory writes
Box SAD	5	6	3
Software population count	4	5	1
Hardware population count	2	1	1

A definitive advantage of the census transform-based matching is that additional disparity estimates are quite fast to compute compared to the competing SAD method. Table 4.4 lists the operation counts needed to compare one neighborhood of one pixel to one correspondence candidate. Although the lookup table -based

comparison is not that far from SAD in terms of the operation count, the difference between their processing speeds can be explained by the more cache-friendly memory access patterns of the transform-based methods. This is demonstrated in Figure 4.8. The data for the below figures have been computed using pure C implementations running on a desktop PC. Hardware assisted population count was not used, which would improve the scaling of census even further.

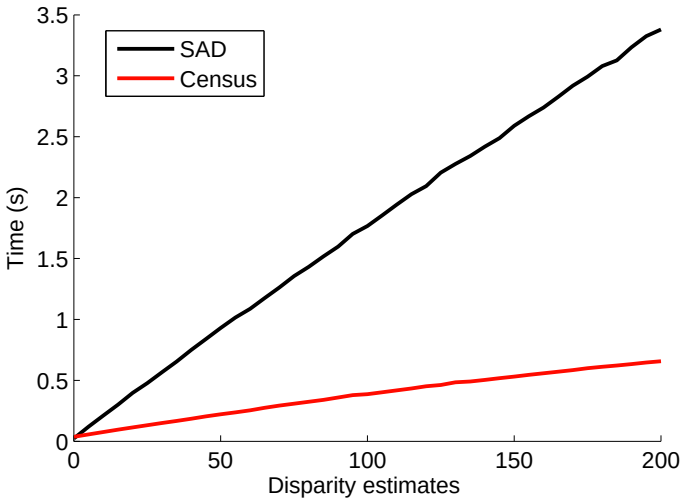


Figure 4.8: Processing time scaling as a function of the number of disparity estimates for O(1) aggregation SAD and the census transform-based matching algorithms

As could be expected, both SAD and transform-based methods scale linearly as a function of the image size of the source stereo pair (Figure 4.9) Also unsurprisingly, census matching maintains its speed advantage over SAD when the image size increases.

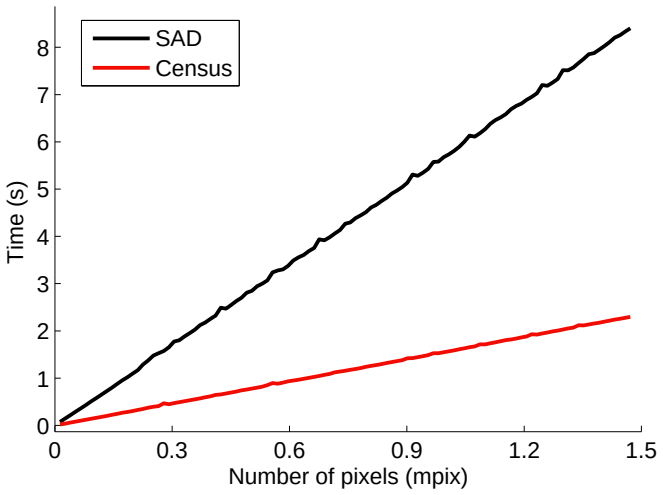


Figure 4.9: Processing time scaling as a function of the image size for O(1) aggregation SAD and the census transform-based matching algorithms over 50 disparity estimates.

The initial results quality-wise do not favor the census, but the trivial computation allowed by its simplicity is a definitive benefit. If cost volume aggregation is applied, the results of census and the simplest matrix multiplication-based transform, the simplified Haar provide very similar results. If there is no aggregation applied, the result of the matching has a large number of mismatches. Fortunately though, the correct matches are still dominant in the result, so it can be quite effectively corrected with post-processing. Some loss of detail can not be avoided, though, as the filtering window necessary for this will be relatively large.

4.2.2 Subjective testing

The traditional metrics were noticed to be quite unreliable in the case of census-based matching. Due to the difficulties in finding such objective quality metrics that would appropriately portray not only the reconstruction accuracy, but also the visual quality of virtual views rendered using the depth, a small scale subjective test was organized. 20 test subjects were shown sequentially a series of 16 image pairs. Two image pairs were used for training and control, and were artificially degraded using photo manipulation software. The rest were shown in a randomized order (both the order of image pairs and the order of images inside the pair). The other image in the pair was always a rendered virtual view based on depth acquired with census-based matching, while the other was rendered using the same method, but based on depth estimated with box filter aggregated SAD with window size 11x11.

The subjects were asked to mark down which of the two images looked better, or in the case of lacking distinctive differences, "no difference". Images were viewed on a matte 2D display of a desktop computer in even office lighting. In order to keep the test simple, rendered stereo pairs were not shown on a 3D display, where factors such as each subject's capability of seeing the stereoscopic 3D effect would have greatly influenced the results. Occlusion filling for the rendering was done by filling such pixels that did not get a projected value with a median value of surrounding pixels. Granted it is not the most sophisticated of methods, but both of the compared matching algorithms get treated the same way.

Looking at the results that favor either SAD or census, SAD does get more votes as providing better quality by 13 percentage units (27% favor census, 40% SAD). The "no difference" answers, however, level the playing field, as 33% of total answers did not distinguish the two from each other. Looking at the distribution of votes for SAD (Figure 4.10), certain image pairs have a huge bias favoring it. Further analysis of the images behind the vote distributions shows that few images have some very distinctive and specific distortions that are likely to be the cause of the results for those pairs. These specific errors were not addressed in this thesis, but analysis of the errors may well lead to post processing methods capable of handling them.

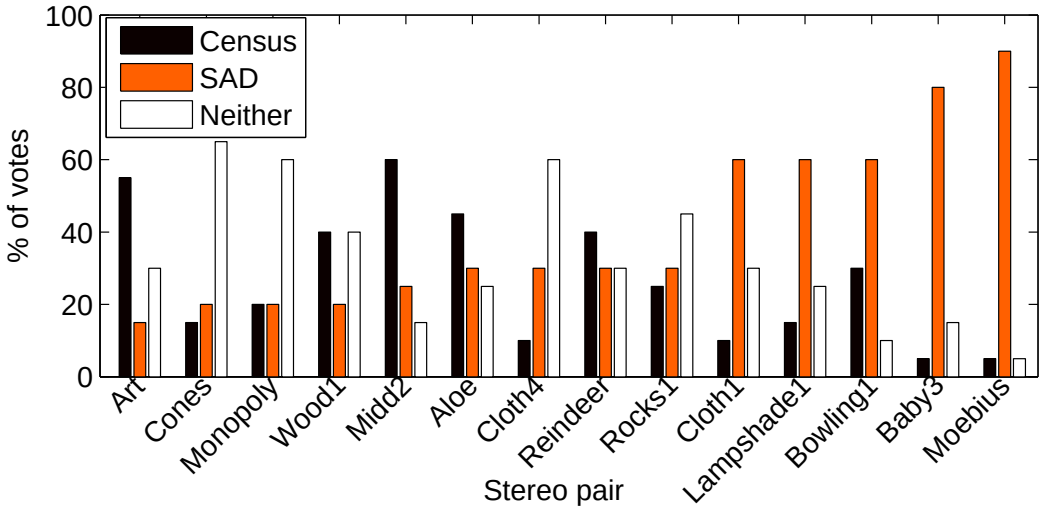


Figure 4.10: The distribution of votes during the subjective test for each stereo pair for choices "census", "SAD" and "neither"

In Figure 4.11 the two cards leaning against each other is somehow difficult for census to process. A large portion of the cards gets a wrong disparity value, but the error is present in both the left and right disparity estimates, as it is not affected by the left-right consistency check. In the rendered view, the error is shown as an obvious distortion. All though SAD is also having some problems with the same area, the distortion is not so pronounced.

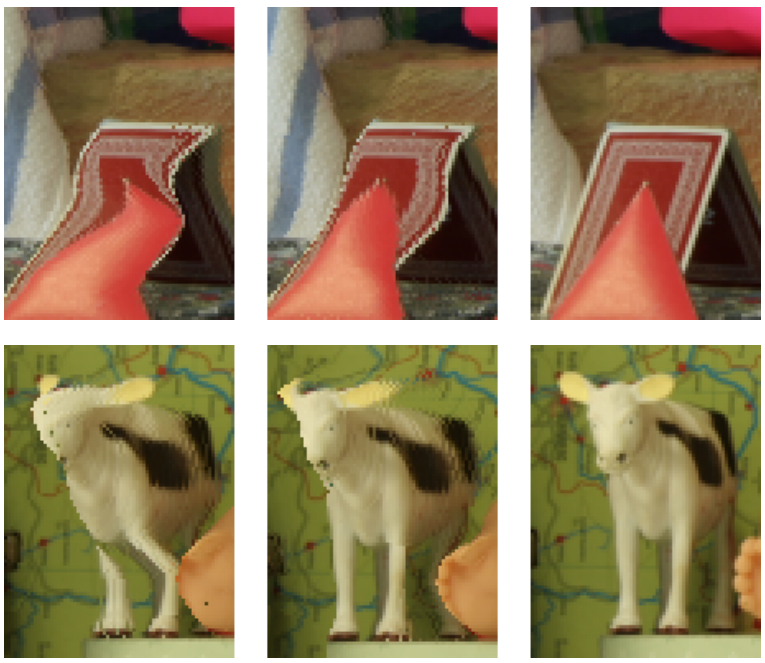


Figure 4.11: Zoomed partitions of views rendered from depth estimated by census (left), SAD (center) and the original view from the same viewpoint. The source stereo pairs are Moebius and Baby3 from [8]

Another kind of problem is in shown in the lower row of Figure 4.11. The two miniature cow toys on both sides of the baby doll are detected by census, but the areas have such a low ratio of correct matches that the shape of those toys is lost in post processing. The toys are large enough not to be completely hidden by the large median filter applied, but they do suffer from it as the resulting disparity map does not correspond to the source image. The appearance of those toys is likely so distinguishable to the human observer that their distortion into something less recognizable is easily noticed.

5. CONCLUSIONS

A novel method of applying transform-based similarity measures to finding stereo correspondences has been presented. In most cases, transform-based metrics offer better quality in comparison to SAD with box filter -based aggregation. Another benefit is a better computational scalability with the number of disparity measures. This is due to the more compact presentation of the required information, which better utilizes the memory architecture of modern processors and the ability to process large segments of that presentation at once even without utilizing parallelization via dedicated SIMD accelerators. The noise of the resulting disparity characteristic to transform-based methods map is shown to be controllable using simple post processing to match and surpass the quality of the reference metric.

The speed gain in comparison to SAD comes from the lack of a separate aggregation step, where the cost volume is filtered using spatial filters. Transform-based matching methods offer the benefit of having a constant amount of work to be done in the beginning, and a lower coefficient on the linear component of the computational complexity with regards to number of disparity estimates. A drawback is the $O(n^2)$ complexity with regards to the window radius, and the constant window of size 8×8 has limited performance on large source image sizes. The 8×8 window size utilizes hardware efficiently, which makes it tricky to change the window size freely and retain the computational benefits. If improved quality is the focus, sacrificing some speed in favor of introducing a separate aggregation step will further increase the quality of transform-based matching.

As far as quality and speed are considered, the Haar wavelet transform is clearly a better choice for most applications. However, DCT is frequently used in other image processing tasks. It may allow for synergy benefits between stereo matching and other algorithms that are also transforming some or all of the same windows. DCT might also have accelerated implementations readily available on some platforms. An integer-based approximation of DCT can work for this purpose even better than the original transform if chosen correctly.

The transform-based methods presented here have an inherent ability to ignore constant differences in intensity between stereo pairs due to the way the constant term is included in a single component, which as the average of the window is always positive anyway. Also the census transform exhibits this property due to the way it

uses the window's center pixel as a reference. This allows changes in the exposure between the stereo pair with no extra computational cost, unlike SAD, where the intensity changes must be compensated by additionally computing mean values for windows.

The approach to evaluating these metrics was to compare them at the basic level as similarity measures for finding stereo correspondences. A modern stereo matching algorithm will also consider confidence metrics and more elaborate schemes for post filtering. There are further studies to be made on this area. Also the potential synergy between denoising based on the same transformed windows as the matching algorithm is a potential continuation of this work. For instance, the efficient finding of correspondences may be well suited for non-local denoising methods which already transform windowed sections of the image into DCT domain. Based on the experimental results, it is reasonable to assume, that the described transform domain approach would work to some degree on any generalized harmonic transforms. Also, it is possible that there exists such transform domain representations that would lead to even better quality. Computationally though, an optimized implementation of the unscaled Haar wavelet transform is hard to beat.

The application of these methods on a mobile platform has been studied and described. Transforms based on matrix multiplication such as DCT offer good quality matching, but the performance constraints of the platform call for even simpler processing. In an effort to make it possible to run stereo matching at reasonable rates without specialized hardware accelerated solutions like a DSP or a GPU, census transform is used instead.

Some objective measures rate the quality of the census matching somewhat below that of SAD-based methods, but on the other hand, some also favor census. Small scale subjective studies imply that the census has some specific problems that observers pay attention to. This tips the scales in favor SAD in the general case, but still, for a large number of tested scenes, census provides similar or better results.

Speed-wise the census shows promising results. It is embarrassingly parallel at multiple levels, offering great possibilities to utilize any parallel processing capabilities offered by the hardware platform. Due to the deficiencies in compiler behavior regarding the SIMD extension of ARM, the experimental implementation does not fully benefit from the potential of the NEON core. A detailed implementation using assembly should be done for any actual implementation in a production environment. Compared to the closest competition from traditional stereo matching, the box filtered SAD, census is faster, especially given there is a specialized population count operator in the hardware platform. The more cache-friendly memory access patterns make it faster even if population count has to be emulated in software. Any post processing done on the disparity image will likely not be affected

by the amount of disparity estimates. Therefore the benefit of having a better scaling with the number of disparity estimates is not affected by the more intensive post-processing required by census results.

A less intrusive method of post processing would greatly improve the reconstruction quality of the census transform, as there are rarely objects that are estimated completely wrong. The bilateral filter can be used to some extent to enforce constraints from the source stereo pair's color information, but the effect of a single application of the filter will not have a very powerful effect. Even the efficient implementations of the bilateral filter are computationally quite intensive in this hardware environment, so iterative approaches will quickly become too heavy. Consistency checking between two alternate disparity maps is a promising option as a post processing tool if it is applied after the disparity maps have been denoised using median filtering. Median filtering can be efficiently applied even with large window sizes using the distributive properties of histograms to compute them for each successive sliding window.

REFERENCES

[1] R. Szeliski, *Computer Vision: Algorithms and Applications.* New York: Springer, 2010.

[2] R. J. Trew and J. E. Brittain, "3D media & displays," *Special issue, Proceedings of the IEEE*, vol. 99, no. 4, 2011.

[3] K. Müller, P. Merkle, and T. Wiegand, "3-D video representation using depth maps," *Proceedings of the IEEE*, vol. 99, no. 4, pp. 643–656, 2011.

[4] S. B. Gokturk, H. Yalcin, and C. Bamji, "A time-of-flight depth sensor - system description, issues and solutions," in *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04. Conference on*, pp. 35–35, 2004.

[5] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pp. 127–136, 2011.

[6] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 807–814 vol. 2, 2005.

[7] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze, "A fast stereo matching algorithm suitable for embedded real-time systems," *Computer Vision and Image Understanding*, vol. 114, pp. 1180–1202, 11 2010.

[8] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1, pp. 7–42, 2002.

[9] J. Weng, P. Cohen, and M. Herniou, "Camera calibration with distortion models and accuracy evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 965–980, 1992.

[10] A. Fusiello, E. Trucco, and A. Verri, "A compact algorithm for rectification of stereo pairs," *Machine Vision and Applications*, vol. 12, no. 1, pp. 16–22, 2000.

[11] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.

[12] J. Banks, M. Bennamoun, and P. Corke, "Non-parametric techniques for fast and robust stereo matching," in *TENCON '97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications., Proceedings of IEEE*, vol. 1, pp. 365–368 vol.1, 1997.

[13] F. Tombari, S. Mattoccia, L. D. Stefano, and E. Addimanda, "Classification and evaluation of cost aggregation methods for stereo correspondence," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, 2008.

[14] F. C. Crow, "Summed-area tables for texture mapping," *SIGGRAPH Comput.Graph.*, vol. 18, pp. 207–212, jan 1984.

[15] A. Ansar, A. Castano, and L. Matthies, "Enhanced real-time stereo using bilateral filtering," in *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on*, pp. 455–462, 2004.

[16] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision, 1998. Sixth International Conference on*, pp. 839–846, 1998.

[17] Q. Yang, K. H. Tan, and N. Ahuja, "Real-time O(1) bilateral filtering.," in *CVPR*, pp. 557–564, 2009.

[18] S. Gordon, "Simplified use of 8x8 transforms - updated proposal & results," tech. rep., Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, 2004.

[19] D. V. Papadimitriou and T. J. Dennis, "Stereo disparity analysis using phase correlation," *Electronics Letters*, vol. 30, no. 18, pp. 1475–1477, 1994.

[20] M. A. Muquit, T. Shibahara, and T. Aoki, "A high-accuracy passive 3D measurement system using phase-based image matching," *IEICE Trans.Fundam.Electron.Commun.Comput.Sci.*, vol. E89-A, pp. 686–697, mar 2006.

[21] T. Shibahara, T. Aoki, H. Nakajima, and K. Kobayashi, "A sub-pixel stereo correspondence technique based on 1D phase-only correlation," in *Image Processing, 2007. ICIP 2007. IEEE International Conference on*, vol. 5, pp. 221–224, 2007.

[22] C. D. Kuglin and D. C. Hines, "The phase correlation image alignment method," *IEEE Conference on Cybernetics and Society*, pp. 163–165, 1975.

[23] W. H. Chen, C. Smith, and S. Fralick, "A fast computational algorithm for the discrete cosine transform," *Communications, IEEE Transactions on*, vol. 25, no. 9, pp. 1004–1009, 1977.

[24] K. R. Rao, P. Yip, and V. Britanak, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Orlando, FL, USA: Academic Press, Inc, 2007.

[25] H. Qi, W. Gao, S. Ma, and D. Zhao, "Adaptive block-size transform based on extended integer 8x8/4x4 transforms for H.264/AVC," in *Image Processing, 2006 IEEE International Conference on*, pp. 1341–1344, 2006.

[26] N. Ahmed, K. Rao, and A. Abdussattar, "BIFORE or Hadamard transform," *Audio and Electroacoustics, IEEE Transactions on*, vol. 19, no. 3, pp. 225–234, 1971.

[27] S. Agaian, H. Sarukhanyan, K. Egiazarian, and J. Astola, *Hadamard Transforms*. SPIE Press, 1 ed., 2011.

[28] R. S. Stankovic and B. J. Falkowski, "The Haar wavelet transform: its status and achievements," *Computers & Electrical Engineering*, vol. 29, pp. 25–44, 1 2003.

[29] C. A. Wilson and J. A. Theriot, "A correlation-based approach to calculate rotation and translation of moving cells," *Image Processing, IEEE Transactions on*, vol. 15, no. 7, pp. 1939–1951, 2006.

[30] O. Urhan, M. K. Gullu, and S. Erturk, "Modified phase-correlation based robust hard-cut detection with application to archive film," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 16, no. 6, pp. 753–770, 2006.

[31] K. Ito, A. Morita, T. Aoki, T. Higuchi, H. Nakajima, and K. Kobayashi, "A fingerprint recognition algorithm using phase-based image matching for low-quality fingerprints," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 2, pp. II–33–6, 2005.

[32] K. Miyazawa, K. Ito, T. Aoki, K. Kobayashi, and H. Nakajima, "An efficient iris recognition algorithm using phase-based image matching," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 2, pp. II–49–52, 2005.

[33] I. Ito and H. Kiya, "DCT sign-only correlation with application to image matching and the relationship with phase-only correlation," in *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, vol. 1, pp. I–1237–I–1240, 2007.

[34] S. E. Anderson, "Bit twiddling hacks," 2012. [Online] Cited 25.04.2012 http://graphics.stanford.edu/~seander/bithacks.html.

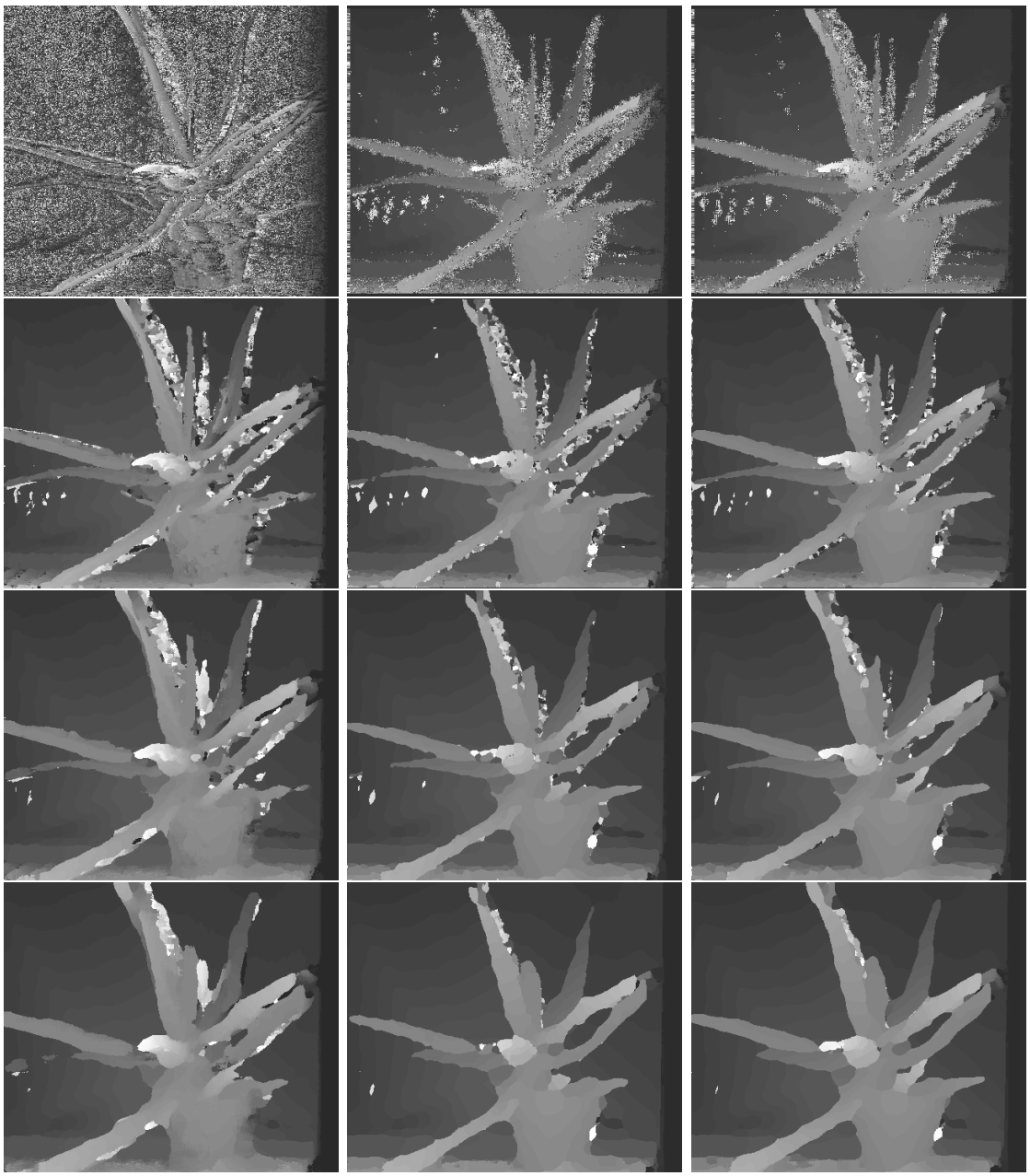
[35] Intel Corporation, "Intel 64 and IA-32 architectures - software developer's manual," 2011.

[36] ARM, "Cortex A8 technical reference manual," 2006.

[37] Texas Instruments, "OMAP 3 architecture from TI for MID," 2008.

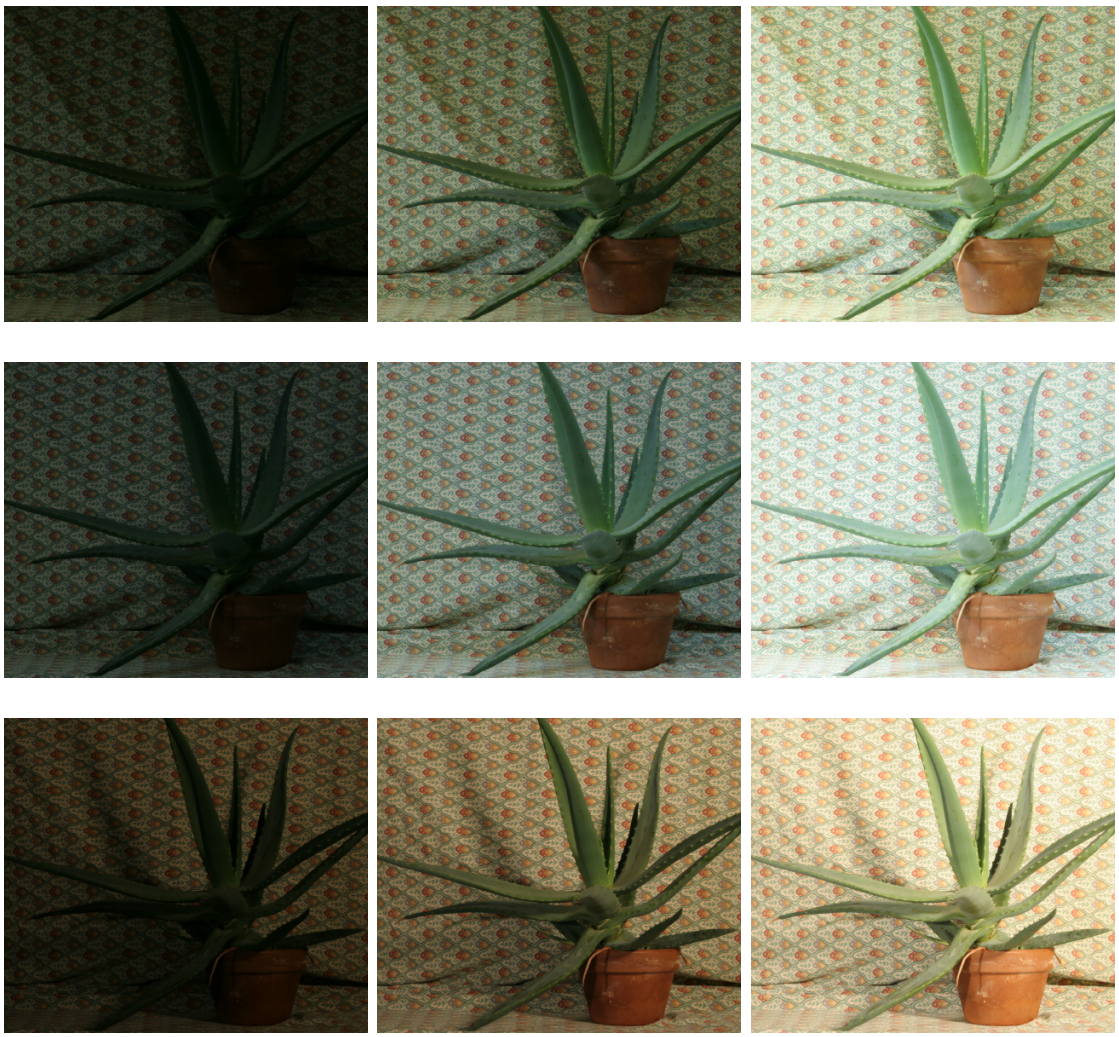
[38] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pp. 1–8, 2007.

APPENDIX 1: AGGREGATED DISPARITY MAPS



From left to right: SAD, DCT, Haar and from top to bottom, box aggregation radii 0, 2, 4, 6 and 9

APPENDIX 2: EXAMPLE DATA: VARYING ILLUMINATION AND EXPOSURE



Example (the Aloe stereo pair) from the data set used for comparing the performance under different lighting conditions and with different exposure times. The exposure varies from left to right (exp0...2 as referred to in the results section), while the illumination changes from top to bottom (Illum 1...3).